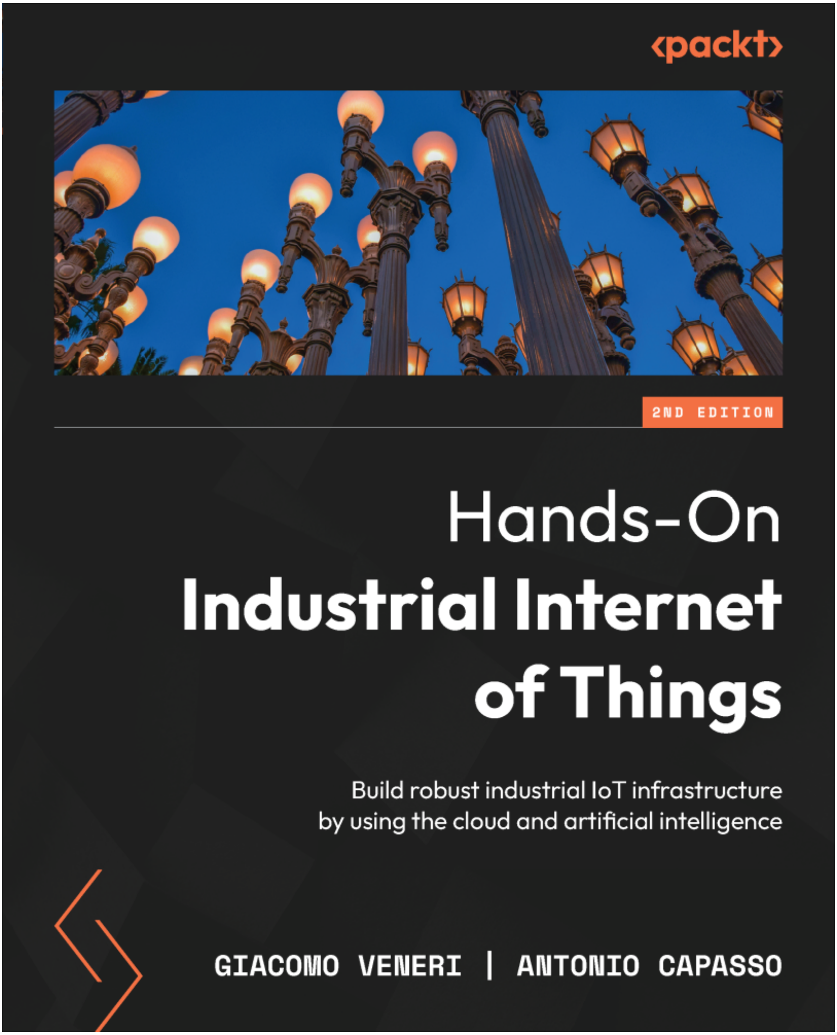


packt

2ND EDITION

Hands-On Industrial Internet of Things

Build robust industrial IoT infrastructure by using the cloud and artificial intelligence

GIACOMO VENERI | ANTONIO CAPASSO

Hands-On Industrial Internet of Things

Copyright © 2024 Packt Publishing

All rights reserved . No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

The author acknowledges the use of cutting-edge AI, in this case Perplexity AI, with the sole aim of enhancing the language and clarity within the book, thereby ensuring a smooth reading experience for readers. It's important to note that the content itself

has been crafted by the author and edited by a professional publishing team.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the authors, nor Packt Publishing or its dealers and distributors, will be held liable for any damages caused or alleged to have been caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

Group Product Manager : Preet Ahuja

Publishing Product Manager : Suwarna Rajput

Book Project Manager : Ashwin Kharwa

Senior Editors : Mudita Sonar, Runcil Rebello

Technical Editor : Irfa Ansari

Copy Editor : Safis Editing

Proofreader : Runcil Rebello

Indexer : Rekha Nair

Production Designer : Prashant Ghare

DevRel Marketing Coordinator : Rohan Dobhal

First published: November 2018

Second edition: November 2024

Production reference: 1291024

Published by Packt Publishing Ltd.

Grosvenor House

11 St Paul's Square

Birmingham

B3 1RB, UK

ISBN 978-1-83588-746-2

www.packtpub.com

To my wife, Stefania. My lovable everything.

– Giacomo Veneri

Contributors

About the authors

Giacomo Veneri was born in Siena, Italy, in 1973. He is an expert in data processing and industrial internet of things (IIoT). Working actively as an AI and IoT manager, he is the author of several books and scientific articles. He graduated from the University of Siena with a degree in computer science in 1999 and graduated in 2014 with a PhD in neuroscience and neural computation. He has achieved more than 10 certifications in programming, AI, and the cloud. Today, Giacomo is an AI director.

Antonio Capasso graduated from the University of Naples with a degree in computer automation in 1999 and a degree in computer science in 2003. He has been working for twenty years on large and complex IT projects related to the industrial world in a variety of fields (automotive, pharma, food and beverage, and oil and gas), in a variety of roles (programmer, analyst, architect, and team leader) with different technologies and software. Since 2011, he has been involved in building and securing IIoT infrastructure. He currently lives in Tuscany, where he enjoys trekking and swimming.

About the reviewers

Balaji Kannapan has 25 years of overall experience and 20 years in the SAP supply chain across the life science, healthcare, automotive, consumer goods, manufacturing, IS oil, chemicals, and mining industries. He has accomplished aligning technology solutions integrated with various applications with strategic business requirements for a wide range of companies. He received a Bachelor of Engineering and a Master of Business Administration from the University of Madras. He has co-authored a book on warehouse management.

I'd like to thank my family and friends who understand the time and commitment it takes to write and review books. Contributing to this field would not be possible without attaining knowledge in consulting over the last several years. I am deeply indebted to the publishing companies for having the necessary framework for peer review and the people running them.

Diwakar Reddy Peddinti is a seasoned software engineer with over 12 years of diverse experience in domains such as embedded systems, aviation, logistics, and IIoT. He holds a master's degree in data science and currently excels as a tech lead at a pioneering IIoT start-up in Silicon Valley. Diwakar's extensive expertise encompasses designing enterprise-grade mobile applications, developing robust backend systems, performing data analytics on vast IoT datasets, and creating innovative machine learning and deep learning models. With a commitment to advancing technology and a proven track record in his field, Diwakar is dedicated to pushing the boundaries of what is possible in the ever-evolving software engineering landscape.

Somasundaram Palaniappan is a skilled artificial intelligence and machine learning engineer, specializing in the development and deployment of advanced machine learning models. He has a strong background in software development, complemented by a post-graduate program in cloud computing from Great Lakes Executive Learning (2022-2023), a post-graduate program in artificial intelligence and machine learning from Texas McCombs, The University of Texas at Austin (2021-2022), and a Bachelor's Degree in petroleum engineering from the University of Petroleum and Energy Studies, Dehradun (2010-2014). Currently, he serves as a lead AI engineer at Baker Hughes, where he applies his expertise to drive innovation and cutting-edge solutions.

I would like to extend my sincere gratitude to the *author for entrusting me with the opportunity to review this work. It has been an enriching experience to engage with the depth and insight presented throughout the book. I am also thankful to the publishing team for their support and collaboration during the review process. Special thanks to my family, friends, and colleagues for their constant encouragement and feedback.*

Table of Contents

Preface

Part 1: Industrial IoT

1

Introduction to Industrial IoT

Technical requirements

IoT background

History and definition

IoT enabling factors

IoT use cases

IoT key technologies

What is the IIoT?

Use cases of the IIoT

IoT and IIoT – similarities and differences

IoT analytics, AI, and AIoT

Industry environments and scenarios covered by IIoT

Summary

2

Understanding the Industrial Process and Devices

Technical requirements

The industrial process

Automation in the industrial process

Control and measurement systems

Types of industrial processes

The CIM pyramid

CIM pyramid architecture – devices and networks

CIM networks

The IIoT data flow

The Industrial IoT data flow in a factory

The edge device

The Industrial IoT data flow in the cloud

Summary

3

Industrial Data Flow and Devices

Technical requirements

The IIoT data flow in the factory

Measurements and the actuator chain

Sensors

Converters

Actuators

Controllers

Microcontrollers

PLCs

Advanced Control

DCS

Industrial protocols

The OSI/ISO model

Automation networks

The fieldbus

Supervisory control and data acquisition (SCADA)

Historian

ERP and MES

The asset model

ISA-95 equipment entities

SA-88 extensions

Summary

4

Implementing the Industrial IoT Data Flow

Discovering OPC

OPC Classic

OPC UA

The OPC UA information model

OPC UA sessions

The OPC UA security model

The OPC UA data exchange

OPC UA notifications

Understanding the IIoT edge

The IoT edge versus the IIoT edge

The fog versus the IIoT edge

The edge architecture

The edge gateway

The edge computing

The edge tools

Edge implementations

Azure IoT Edge

AWS IOT Edge

Node-RED

Edge internet protocols

Implementing the IIoT data flow

IIoT data sources and data gathering

PLC

DCS

SCADA

Historians

Edge deployment and data flow scenarios

The edge on a fieldbus setup

The edge on OPC DCOM

The edge on OPC-Proxy

The edge on OPC UA

OPC UA on the controller

Summary

5

Applying Cybersecurity

What is a DiD strategy?

Making people aware

Understanding technology

Operating modes and procedures

The DiD in an industrial control system environment

Firewalls

Common control-network-segregation

architectures

Network separation with a single firewall

A firewall with a DMZ

A paired firewall with a DMZ

A firewall with DMZ and VLAN

Securing the IIoT data flow

Securing the edge on a fieldbus

Securing the edge on OPC DCOM

Securing the edge on OPC Proxy

Securing the edge on OPC UA

Securing OPC UA on a controller

Summary

6

Performing an Exercise Based on Industrial Protocols and Standards

Technical requirements

The OPC UA Simulation Server

OPC UA Server Node.js

Prosys OPC UA Simulator – optional step

The edge

Installing a simple edge client

The Node-RED edge

Node-RED

Summary

Part 2: Industrial IoT Architecture

7

Developing Industrial IoT and Architecture

Technical requirements

Introduction to the IIoT platform and architectures

Microservice, containers, and serverless computing

Docker

Microservice-oriented technologies

Software as a service

The standard IIoT flow

Understanding the time-series technologies

Not only sensors

Asset registry

Data processing and the analytics platform

Excursion monitoring analytics

Advanced analytics

Big data analytics

Hands-on with InfluxDB, Node-RED, and OPC UA

Using Docker Compose and Docker Build

Summary

8

Implementing a Custom Industrial IoT Platform

Technical requirements

An open source platform in practice

Mosquitto as a data gateway

Storing time-series data on InfluxDB

Connecting InfluxDb and MQTT Data Broker

Starting and testing the data flow

Visualizing data using Grafana

Developing our batch analytics with Airflow

Developing our online analytics with

Airflow

What’s still missing here?

Building an asset registry to store asset information

Building an asset model with Neo4j

Pros and cons of the proposed platform

Other technologies

Other open source technologies for analytics

Other open source platforms

Other IIoT data beyond the time series

Summary

9

Building an AWS Industrial IoT Solution

Technical requirements

Why do we use commercial IIoT platforms?

AWS architecture

AWS IoT

Registering with AWS

Installing the AWS command-line interface client

IoT Core

Identifying the AWS MQTT endpoint

Connecting Node-RED

Using the AWS SDK

Greengrass V2

SiteWise

Summary

10

Implementing an Industrial IOT Data Flow with AWS

Technical requirements

Architecture of the exercise

AWS IoT Analytics

SageMaker and Athena

IoT Analytics

Storing data

Visualization

Workforce users

Grafana

About visualization

Summary

11

Performing a Practical Industrial IoT Solution with Azure

Technical requirements

Azure IoT ecosystem

Registering for Azure IoT

Azure IoT Hub

Registering a new device

Building a custom edge

Azure IoT Edge

Device twins

Monitoring messages sent through IoT Hub Metrics

Showing messages sent through Service Bus

Summary

12

Implementing an Industrial IoT Data Flow with Azure

Technical requirements

Architecture of the proposed exercise

Setting the data flow

Data storage

Data processing

Azure Synapse

Machine Learning analytics

Building visualizations

Azure Data Explorer

Grafana

Power BI

IoT Central

More on Azure Data Explorer

Comparing the platforms

Summary

Part 3: Industrial Artificial Intelligence of Things

13

Performing Diagnostic, Maintenance, and Predictive Analytics

Technical requirements

Jupyter

IIoT analytics

The different classes of analytics

Descriptive analytics

Diagnostic analytics

Predictive analytics

Prescriptive analytics

IIoT analytics technologies

Rule-based analytics

Model-based analytics

Building IIoT analytics

Step 0 – problem statement

Step 1 – dataset acquisition

Step 2 – EDA

Step 3 – building the model

Step 4 – packaging and deploying (MLOps)

Step 5 – monitoring

Understanding the role of the infrastructure

Deploying analytics

Streaming versus batch analytics

Condition-based analytics

Interactive analytics

Analytics on the cloud

Analytics on the edge

Analytics on the controller

Advanced analytics

OSA

Analytics in practice

Anomaly detection in practice

Anomaly detection with unsupervised ML

Anomaly detection with supervised ML

Predictive production analytics in practice

Prescription using language models

Summary

14

Implementing a Digital Twin – Advanced Analytics

Technical requirements

Digital twins

Digital twins in practice

Implementing a digital twin in IIoT

AI and IoT

ML

DL

GenAI

AGI

Developing a digital twin

Preparing the development environment

Evaluating the RUL of 100 engines

Monitoring a wind turbine

Platforms for digital twins

Digital twins platforms

Advanced modeling

Other kinds of IIoT data

Summary

15

Deploying an Analytics Model

Technical requirements

Understanding model-as-a-service

Developing our first MaaS with MLflow

Starting MLflow

Developing our ML model

Understanding Airflow integration

Working with the Azure ML service

Starting with the Azure ML service

Understanding the Azure ML workspace

Developing wind turbine digital twins with Azure ML

Cleaning the resources

IoT Hub integration

Azure IoT Edge

Implementing analytics on Amazon SageMaker

Accessing SageMaker Studio

Preparing an S3 bucket to store data

Developing a digital twin with SageMaker

Consuming the model from AWS IoT Core

Understanding the advanced features of SageMaker

Understanding GCP and multi-cloud solutions

Summary

Index

Other Books You May Enjoy

Preface

In today's automation-driven era, precision is crucial, and the Industrial Internet of Things (IIoT) has a remarkable impact. This updated guide offers essential knowledge for establishing remote-access networks and explores the technologies fueling the IIoT revolution.

Written by an IIoT and AI expert, this book serves as a comprehensive resource for enhancing skills in emerging technologies, introducing new techniques from Azure and AWS, and allowing you to stay up to date with the latest advancements in the field. You'll dive into AIoT and MLOps applied to IIoT, equipping yourself to handle complex projects confidently. The book covers identifying and connecting industrial data sources from various sensors, advancing from foundational concepts to professional skills. You'll learn how to connect these sensors to cloud networks such as AWS IoT, Azure IoT, and open source IoT platforms, and extract data from the cloud to your devices. Through hands-on experience with tools such as Node-RED, OPC-UA, MQTT, NoSQL, defense in depth, and Python, you'll develop streaming and batch-based AI algorithms.

By the end, you'll have achieved a professional level of expertise in the cloud, IoT, and AI, enabling you to build more robust, efficient, and reliable IoT infrastructure in your industry.

Who this book is for

If you are an IoT architect, developer, AI engineer, or a stakeholder involved in the architectural aspects of the IIoT, this book is tailored for you. The only prerequisites are a solid understanding of programming languages and networking concepts.

What this book covers

Chapter 1 , *Introduction to Industrial IOT* , provides some background to industrial IoT, the story, use cases, and the contrast with the home internet of things. It helps you understand the Industrial IoT Process

Chapter 2 , *Understanding the Industrial Process and Devices* , helps you understand the industrial process and devices and defines the factory processes. This chapter describes the concept of a distributed control system (DCS), programmable logic controllers (PLCs), supervisory control and data acquisition (SCADA), Historian, manufacturing execution system (MES), enterprise resource planning (ERP), and fieldbus. It introduces the International Electrotechnical Commission (IEC)-61131 and the CIM pyramid. Finally, it designs a big picture, from equipment through to the cloud.

Chapter 3 , *Industrial Data Flow and Devices* , details which equipment, devices, network protocols, and software layers manage the industrial IoT data flow along its path, from the sensors on the factory floor to the edge that is the external boundary of the industrial IoT data flow inside the factory.

Chapter 4 , *Implementing the Industrial IoT Data Flow* , explains how to implement the industrial IoT data flow in a complex industrial plant. This journey starts with an understanding of how to select the industrial data source to connect to for the purpose of gathering the data and ends by providing five network scenarios for edge deployment in industrial plants.

Chapter 5 , *Applying Cybersecurity* , explores the industrial IoT data flow from the cybersecurity perspective, outlining the goals of the DiD strategy, and the most common network architecture to secure industrial control systems, including the five network scenarios for edge deployment discussed in the previous chapter.

Chapter 6 , *Performing an Exercise Based on Industrial Protocols and Standards* , explores how to implement a basic data flow from the edge to the cloud by means of OPC UA and Node-RED.

Chapter 7 , *Developing Industrial IoT and Architecture* , outlines the basic concepts regarding industrial IoT data processing, providing the key principles for storing time series data, handling the asset data model, processing the data with analytics, and building digital twins.

Chapter 8 , *Implementing a Custom Industrial IoT Platform* , shows how to implement a custom platform leveraging the most popular open source technologies: Node.js, Docker, InfluxDB, Neo4J, Apache Airflow, Mosquitto, and Docker.

Chapter 9 , *Building an AWS Industrial IOT Solution* , explores the solutions proposed by Amazon Web

Services (AWS) and the capabilities of the AWS IoT platform. This chapter introduces the Edge IoT of AWS (Greengrass), the IoT Core, and AWS SiteWise and builds the first part of the proposed architecture. We will learn about these technologies by performing a practical exercise.

Chapter 10 , *Implementing an Industrial IOT Data Flow with AWS* , concludes the proposed architecture by implementing a data flow using DynamoDB, AWS analytics, and Grafana to display data. We will gain practical experience with these technologies through hands-on exercise.

Chapter 11 , *Performing a Practical Industrial IoT Solution with Azure* , develops a wing-to-wing industrial IoT solution leveraging Azure, Azure Edge, and the Azure IoT platform. We will learn about these technologies by performing a practical exercise.

Chapter 12 , *Implementing an Industrial IoT Data*

Flow with Azure , finalizes the proposed architecture, implementing the data flow with Azure Cosmos DB, Stream Analytics, and Synapse. We will learn about these technologies by performing a practical exercise.

Chapter 13 , *Performing Diagnostic, Maintenance, and Predictive Analytics* , introduces the basic concepts of analytics and data consumption. It also develops basic analytics for anomaly detection and prediction. The chapter also introduces new concepts of generative AI.

Chapter 14 , *Implementing a Digital Twin – Advanced Analytics* , develops a physics-based and data-driven digital equipment model to monitor assets and systems.

Chapter 15 , *Deploying an Analytics Model* , shows how to develop analytics on Azure ML and AWS SageMaker. Finally, the chapter explores other common technologies.

To get the most out of this book

You will need a version of Python, Docker, and Git installed on your computer. All code has been tested on Python 3.11, Node.js 20, Docker 4.34, Linux, and Windows.

Software/hardware covered in the book	Operating system requirements
Python 3.11+	Windows or Linux
Node.js 20+	
Docker or Docker Community Edition	
Git	

If you are using the digital version of this book, we advise you to type the code yourself or access the code from the book's GitHub repository (a link is available in the next section). Doing so will help you avoid any potential errors related to the copying and pasting of code.

Download the example code files

You can download the example code files for this book from GitHub at https://github.com/PacktPublishing/Hands-On-Industrial-Internet-of-Things-Second-Edition If there's an update to the code, it will be updated in the GitHub repository.

We also have other code bundles from our rich catalog of books and videos available at https://github.com/PacktPublishing/ . Check them out!

Conventions used

There are a number of text conventions used throughout this book.

`Code in text` : Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. Here is an example: "Create a new file called `docker-compose-python.yml` and add the following two services."

A block of code is set as follows:

```
if __name__ == '__main__':
    asyncio.run(main())
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
if __name__ == '__main__':
    asyncio.run(main())
```

Any command-line input or output is written as follows:

```
pip install opcua crypto cryptography
azure-iot-device
```

Bold : Indicates a new term, an important word, or words that you see onscreen. For instance, words in menus or dialog boxes appear in bold . Here is an example: “Lastly, you can also check in IoT Hub, from Devices | device1 | Modules .”

> ***Tips or important notes***

> *Appear like this.*

Get in touch

Feedback from our readers is always welcome.

General feedback : If you have questions about any aspect of this book, email us at customercare@packtpub.com and mention the book title in the subject of your message.

Errata : Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you would report this to us. Please visit www.packtpub.com/support/errata and fill in the form.

Piracy : If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at copyright@packt.com with a link to the material.

If you are interested in becoming an author : If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit authors.packtpub.com .

Share Your Thoughts

Once you've read *Hands-On Industrial Internet of Things* , we'd love to hear your thoughts! Please click here to go straight to the Amazon review page for this book and share your feedback.

Your review is important to us and the tech community and will help us make sure we're delivering excellent quality content.

Download a free PDF copy of this book

Thanks for purchasing this book!

Do you like to read on the go but are unable to carry your print books everywhere?

Is your eBook purchase not compatible with the device of your choice?

Don’t worry, now with every Packt book you get a DRM-free PDF version of that book at no cost.

Read anywhere, any place, on any device. Search, copy, and paste code from your favorite technical books directly into your application.

The perks don’t stop there, you can get exclusive access to discounts, newsletters, and great free content in your inbox daily

Follow these simple steps to get the benefits:

1. Scan the QR code or visit the link below

https://packt.link/free-ebook/978-1-83588-746-2

1. Submit your proof of purchase
2. That's it! We'll send your free PDF and other benefits to your email directly

Part 1:Industrial IoT

This part introduces IoT and industrial IoT, explains industrial processes and protocols, and introduces Edge, MES, SCADA, PLC, DCS, and standards. In the last chapter of this part, we will gain practical experience with these technologies through hands-on exercises.

This part includes the following chapters:

- *Chapter 1 , Introduction to Industrial IoT*
- *Chapter 2 , Understanding the Industrial Process and Devices*
- *Chapter 3 , Industrial Data Flow and Devices*
- *Chapter 4 , Implementing the Industrial IoT Data Flow*
- *Chapter 5 , Applying Cybersecurity*
- *Chapter 6 , Performing an Exercise Based on Industrial Protocols and Standards*

1

Introduction to Industrial IoT

This chapter introduces the reader to the IoT world, highlighting the key factors driving its growth and the main technologies behind it. We will go through the basic concepts of the IoT and how these have been applied, tailored, and specialized to fit Industrial Internet of Things (IIoT) scenarios. We will then look at the similarities and differences between the IoT and the IIoT, and consider some of the key use cases and expected outcomes of the IIoT. The reader will become familiar with some of the key concepts related to the IIoT, such as operational efficiency, preventive and predictive maintenance, and cost optimization. In this chapter, we will also clarify the kind of IIoT that will be dealt with in this book.

We will cover the following topics:

- IoT background
- IoT key technologies
- What is the IIoT?
- Use cases of the IIoT and IoT—similarities and differences
- IoT analytics and Artificial Intelligence of Things (AIoT)
- Industrial environments and scenarios involving IIoT

Technical requirements

In this book, we will work with several open source and proprietary technologies. To simplify the tests that we will carry out, we will use Docker to deploy databases and frameworks.

IoT background

IoT is the most recent stage in the evolution of the internet: from Internet of Content , through Internet of Services (Web 2.0), to IoT:

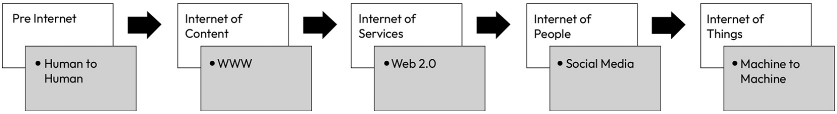

Pre Internet
Human to Human
Internet of Content
WWW
Internet of Services
Web 2.0
Internet of People
Social Media
Internet of Things
Machine to Machine

Figure 1.1 – Evolution of the internet: from social to machine-to-machine

In the last five years, IoT has reached widespread diffusion and a high level of maturity; with over 15 billion devices installed, the IoT has entered our homes, offices, and industries.

This is because the IoT is more than just a new technology that impacts a restricted range of people or a specific market. It can be better understood as a set of technologies that impacts us all, and will change markets, even creating new ones. The IoT is changing our lives, feelings, and perceptions of the physical world daily, by modifying how we interact with it. The development of the IoT is a crucial moment in the history of humanity because it is changing our mindset, culture, and way of living.

You might currently be able to regulate your heating system remotely, but if it lived in the cloud and received information from your car, your calendar, your geolocation, and the weather, then your heating system would be able to regulate itself. When an object lives in the cloud and interacts with other digital images in a web of Artificial Intelligence (AI), that object becomes a smart object.

These developments might seem to be paving the way for a new and perfect world, but there is a dark side to the IoT as well. A lot of personal data and information is stored in the cloud, such that AI can extrapolate information about us and profile our behaviors and preferences. From a different perspective, therefore, the cloud could also be seen as a sort of Big Brother, as in George Orwell's novel *1984* . There is the possibility that our data and profiles could be used not just to enhance our lifestyles but also for more malicious purposes, such as political or economic influence on a large scale.

In this book, we will learn how to implement an end-to-end IoT for industry and how to leverage the IoT data to build AI analytics.

History and definition

The IoT as a concept wasn't officially named until 1999. One of the first examples of the IoT was a Coca-Cola machine, located at Carnegie Mellon University in the early 1980s. Local programmers would connect, through the internet, to the refrigerated appliance, checking to see whether there was a drink available and whether it was cold before making a trip to the appliance.

Kevin Ashton, the executive director of Auto-ID Labs at MIT, was the first to describe the IoT in a presentation for Procter and Gamble. During his 1999 speech, Mr. Ashton stated the following:

> *"Today, computers, and therefore the Internet, are almost wholly dependent on human beings for information. Nearly all of the roughly 50 petabytes (a petabyte is 1,024 terabytes) of data available on the Internet was first captured and created by human beings by typing, pressing a record button, taking a digital picture, or scanning a bar code. The problem is, people have limited time, attention, and accuracy, all of which means they are not very good at capturing data about things in the real world. If we had computers that knew everything there was to know about things, using data they gathered without any help from us, we would be able to track and count everything and greatly reduce waste, loss and cost. We would know when things needed replacing, repairing or recalling and whether they were fresh or past their best."*

Kevin Ashton believed that Radio-Frequency Identification (RFID) was a prerequisite for the

IoT, and that if all devices were tagged, computers could manage, track, and inventory them. In the first decade of the 21st century, several projects were developed to try to implement and translate the IoT philosophy and Ashton's innovative approach into the real world. These first attempts, however, were not so successful. One of the most famous and emblematic cases was the Walmart mandate (2003). By placing RFID tags with embedded circuits and radio antennas on pallets, cases, and even individual packages, Walmart was supposed to be able to reduce inefficiencies in its massive logistics operations and slash out-of-stock incidents, thus boosting same-store sales.

In 2003, Walmart started this pilot project to put RFID tags, carrying electronic product codes, on all pallets and cases involving all its suppliers. In 2009, Procter and Gamble, one of the main suppliers involved in the project, stated that it would exit from the pilot project after validating and checking the benefits of RFID in merchandising and promotional displays.

The unsuccessful story of the Walmart RFID project was caused by various factors:

- Most of the technologies used were in their initial stages of development and their performance was poor. They had sensors with little information and Wi-Fi or LAN connectivity with high power and bandwidth usage.
- The sensors and connectivity devices were expensive due to the small market size.
- There were no common standards for emerging technologies, and there was a lack of interoperability between legacy systems.
- Business cases were not very accurate.
- The technology infrastructure and architecture were organized in vertical silos with legacy hardware and middleware, and a lack of interactions between each silo.
- Technology infrastructure and software architecture were based on a client-server model that still belonged to the so-called second digital platform.

From 2008, several changes were introduced to deal with the preceding issues, which were led mainly by the mobile market. These included the following:

- New higher-performing processors were produced on a large scale at lower cost. These processors supported commercial and/or open operating systems.
- New sensors, which were much more developed, with computation capabilities and high performance, were embedded at a low cost.
- New network and wireless connectivity, which allowed the user to interconnect the devices with each other and to the internet by optimizing bandwidth, power consumption, latency, and range.
- Sensors and devices using Commercial off-the-Shelf (COTS) components were included.
- New power computational capabilities were included.
- Cloud-based, digital platforms were included.

Due to these changes, the IoT evolved into a system that used multiple technologies. These included the internet, wireless communication, micro-electromechanical systems, and embedded systems such as the automation of public buildings, homes, factories, wireless sensor networks, GPS, control systems, and so on.

The IoT consists of any device with an on/off switch that is connected to the internet. If it has an on/off switch, then it can, theoretically, be part of a system.

This includes almost anything you can think of, from cell phones to building maintenance, to the jet engine of an airplane. Medical devices, such as a heart monitor implant or a bio-chip transponder in a farm animal, are also part of the IoT because they can transfer data over a network. The IoT is a large digital network of things and devices connected through the internet. It can also be thought of as a horizontal technology stack, linking the physical world to the digital world. By creating a digital twin of the physical object in the cloud, the IoT makes the object more intelligent thanks to the interaction of the digital twin with the other digital images living in the cloud.

IoT enabling factors

What has changed from the Walmart scenario to make companies support the advent of the IoT? There is no one specific new technology or invention, but rather a set of already existing technologies that have been developed. These have created an ecosystem and a technology environment that makes the connection of different things possible, efficient, easy from a technical perspective, profitable from a market perspective, and attractive from a production-cost perspective.

The technologies that have led to the evolution of IoT ecosystems are as follows:

- A crucial component is new sensors that are much more mature, have more capabilities, and offer high performance at a lower cost. These smart sensors are natively designed to hide the complexity of the signal processing, and they interact easily through a digital interface. The smart sensor is a system itself, with a dedicated chip for signal processing. The hardware for signal processing is embedded in each sensor and miniaturized to the point that it is part of the sensor package. Smart sensors are defined by the IEEE 1451 standard as sensors with small memory and standardized physical connections to enable communication with the processor and the data network. As well as this, smart sensors are a combination of a normal sensor with signal conditioning, embedded algorithms, and a digital interface. The principal catalyst for the growth of smart-sensing technology has been the development of microelectronics at reduced cost. Many silicon manufacturing techniques are now being used to make not only sensor elements but also multilayered sensors and sensor arrays that are able to provide internal compensation and increase reliability. The smart sensors market is expected to reach 100 billion USD by 2027 from 45 billion USD in 2022.

- New networks and wireless connectivity, such as Personal Area Networks (PANs) or Low-Power Networks (LPNs), interconnect sensors and devices to optimize their bandwidth, power consumption, latency, and range. In PANs, several small devices connect directly or through a main device to a LAN, which has access to the internet. Low-Power Wide-Area Networks (LPWANs) are wireless networks designed to allow long-range communications at a low bit rate among battery-operated devices. Their low power, low bit rate, and intended use distinguish these types of networks from the already existing wireless WAN, which is designed to connect users and businesses and carry more data, using more power.
- New processors and microprocessors are now coming from the world of mobile devices. These are very powerful and very cheap. They have produced a new generation of sensors and devices based on standardized and cheap hardware that is driven by open and generic operating systems. These use common software frameworks as an interface, allowing you to transition from a legacy solution, with strictly coupled hardware and software, to a platform built on the COTS component and the adoption of an open software framework.
- The battle of the Real-Time Operating System (RTOS) to gain a larger slice of new markets between the big market players. This places more sophisticated and powerful integrated development platforms at the maker's disposal.
- Virtualization technology divides naturally into the data center, big data, and the cloud. This leads to the following features:
    - CPUs and GPUs, memory, storage, infrastructures,

    platforms, and software frameworks are available as services on demand, with flexible and tailored sizing. These are cheap and available without Capital Expenditure (CAPEX) investment.

    - Elastic repositories are great for storing and analyzing the onslaught of data.
    - The profitable and flexible Operational Expenditure (OPEX) model per CPU, memory, storage, and IT maintenance services is also useful. This creates a business case for migrating the legacy data, infrastructure, and applications to the cloud and making the collection of big data and subsequent analytics possible.

- The convergence of IT and Operational Technology (OT). This has led to the increasing adoption of COTS components in sectors in which hardware was traditionally developed with specific requirements, as is the case in industrial plants.
- The diffusion of mobile devices and social networks has created a culture and a generic mindset with an embedded expectation for the market consumers to encounter the world through an app that shares related information.

The preceding factors are making it possible to transition from a vertical, legacy platform with an application organized hierarchically, with the data confined in silos, to a horizontal, modular, and cloud-based platform. This new platform uses a standardized API layer that provides high interoperability capabilities and the ability to share data and information between applications.

Let's consider what might happen if the Walmart project was carried out now. In 2003, the only RFID technology that existed was active RFID systems. Active RFID systems use battery-powered RFID tags that continuously broadcast their own signals. They provide a long-read range, but they are also expensive and consume a lot of power. Passive RFID systems, on the other hand, use tags with no internal power source and are instead powered by the electromagnetic energy transmitted from an RFID reader. They have a shorter read range, but they are unembeddable, printable, and much cheaper, which makes them a better choice for many industries. Also, at the time of the Walmart project, there were no PANs or LPNs to capture and transmit the label data, meaning the developers had to adopt an expensive, wired connection to transfer the information. The data was then stored in a legacy database and processed by a custom application. If the Walmart project were to be carried out now, instead of in 2003, the tracking information could be carried out by passive RFIDs. The data could be captured by a PAN and transmitted via the cloud to be processed by an application built on top of a common API and framework. This means that all data and information could be easily shared between the project partners.

According to Gartner, the IoT and connected devices market is expected to grow strongly in the next year:

The IT Services for IoT market will represent a 58-billion-dollar opportunity in 2025.

IoT use cases

As previously discussed, the IoT is not just a specific technological innovation, but a radical change that will impact the whole of human society. This means that the IoT will affect nearly every aspect of our personal and professional lives and any sector of the economy, including the following:

- Industrial and manufacturing
- Supply chain
- Retail
- Financial and marketing
- Healthcare
- Transportation and logistics
- Agricultural and environmental
- Energy
- Smart cities and smart grids
- Smart homes and buildings

- Government and military security forces
- Education
- Sports and fitness

All of these are already involved in the digital transformation that has been caused by the IoT and are likely to play a greater role in this in the future. Across all uses of the IoT, the common feature is the smart object.

From a qualitative perspective, a smart object is a multidisciplinary object that includes the following elements:

- The physical product
- Sensors, microprocessors, data storage, controls, and software managed by an embedded operating system
- Wired or wireless connectivity, including interfaces and protocols

This is used to connect the product to its user, all instances of the product to its vendor, or the product to other types of products and external data sources. In another definition, the article *How Smart, Connected Products Are Transforming*

Competition , written by Michael E. Porter and James E. Heppelmann, details four increasing levels that classify the smartness of an object or product:

- Monitoring : This refers to the monitoring of product conditions, external operation, and usage. This enables alerts and notifications of changes.
- Control : Software embedded in the product or in the cloud enables control of product functions, and/or personalization of the user experience.
- Optimization : The previous capabilities are used to create algorithms to optimize the product's operation and use. They enhance product performance, and/or allow features such as predictive diagnostics, service, repair, and so on.
- Autonomy : The combination of the previous capabilities produces a product with autonomous control for self-coordination with other systems and/or self-diagnosis or services.

IoT key technologies

The IoT is sometimes used as a synonym for big data or smart sensors, is sometimes confused with the cloud, and is sometimes linked to machine learning and AI.

All of these things are partially true:

- IoT uses sensors and devices to acquire data
- IoT uses big data technology to store data
- IoT is normally deployed on the cloud to improve scalability
- IoT uses advanced analytics and AI to process data

However, on the flip side, there's this to consider:

- IoT is focused on a data stream, rather than having huge amounts (petabytes) of data storage
- IoT can use on-premises solutions through virtualization technology

In this book, we will explore the most common IoT cloud solutions, such as AWS and Azure. We will also look at other common open source technologies.

These technologies can be used to build an IoT platform from scratch or to integrate with an existing one. We will also consider other commonly used commercial software in the industrial environment. We will discover the new generation of edge computing and the edge gateway, and, finally, we will deal with machine learning and AI.

This journey is also the journey of the IoT from the cloud to the big revolution of general AI expected around 2030:

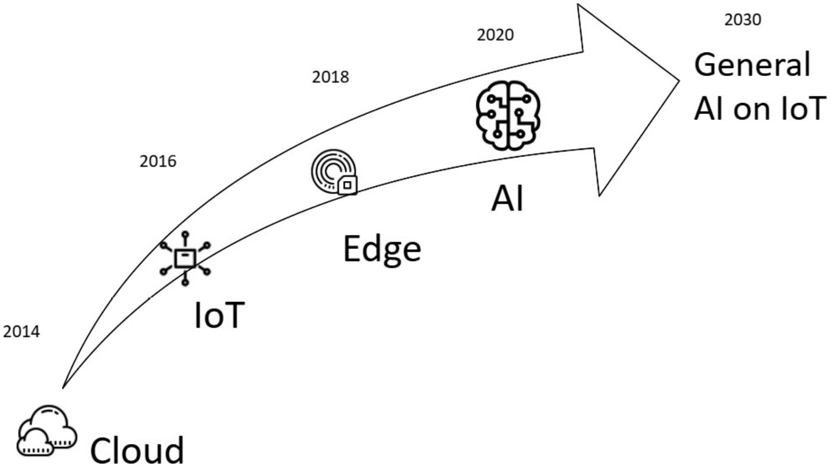

2014
Cloud
2016
IoT
2018
Edge
2020
AI
2030
General AI on IoT

Figure 1.2 – From cloud to AI – the landscape of the evolution of cloud AI and IoT

What is the IIoT?

After the advent of the steam engine in 1760, steam was used to power everything from agriculture to textile manufacturing. This caused the First Industrial Revolution and the age of mechanical production. At the end of the 19th century came the arrival of electricity, new modes of labor organization, and mass production, which started the Second Industrial Revolution. In the second half of the 20th century, the development of semiconductors and the introduction of electronic controllers produced the beginning of the automation era and the Third Industrial Revolution. In the Hannover exhibition of 2011, Henning Kagermann, Wolf-Dieter Lukas, and Wolfgang Wahlster coined the term Industry 4.0 for the project of renewing the German manufacturing system using the capabilities of the latest digital technology.

Industry 4.0 is able to do the following:

- Connect or merge production with information and communication technology
- Merge customer data with machine data
- Harness the capability of machines communicating with machines
- Manage production autonomously in a flexible, efficient, and resource-saving manner

Industry 5.0 is the completion of Industry 4.0, which focuses on sustainability and resilience. Industry 5.0 is a collaborative industry characterized by human-machine cooperation. Industry 5.0 is based on the rapid development of increasingly powerful 4.0 technologies: the IT, AI, and robotics sectors, which are leading to the creation of Cyber Physical Systems (CPSs) and increasingly powerful IoT devices.

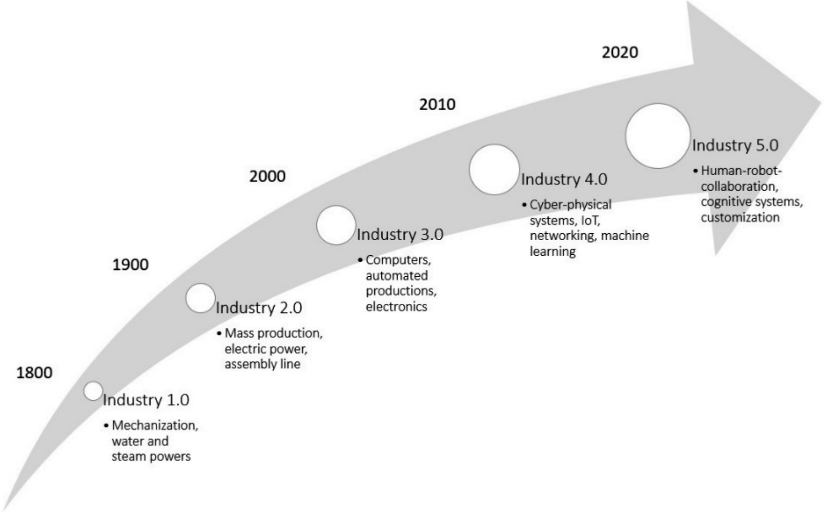

2020
2010
2000
1900
1800
Industry 1.0
• Mechanization, water and steam powers
Industry 2.0
• Mass production, electric power, assembly line
Industry 3.0
• Computers, automated productions, electronics
Industry 4.0
• Cyber-physical systems, IoT, networking, machine learning
Industry 5.0
• Human-robot-collaboration, cognitive systems, customization

Figure 1.3 – From the First Industrial Revolution to the next generation of Industry 5.0

The IoT is almost, by definition, the key to further development of the manufacturing industry by including technologies such as big data analytics, the cloud, robotics, and most importantly, the integration and convergence between IT and OT. Generally speaking, the term IIoT refers to the industrial subset of the IoT. The IIoT, like the IoT, is not just a specific new technology but instead refers to the whole chain of value of a product. Similarly, the IIoT impacts all sectors of the industrial world by significantly modifying the processes at each stage of the life cycle of a product, including how it is designed, made, delivered, sold, and maintained.

Use cases of the IIoT

In few other industries are there so many opportunities to use the IIoT as in manufacturing. In this field, it can be used to connect the physical and the digital, and to build assets such as machines or production and non-production objects. It can also be used to create a variety of product and manufacturing process parameters as part of a vast information network. With manufacturing, we typically tend to think about goods and products, but the bigger opportunity for manufacturers lies in cyber-physical systems, a service economy model, and the opportunities that are presented through exploring data.

The following is a list of several IIoT use cases in manufacturing and their benefits:

- Manufacturing operations : This includes all operations typically performed by the Manufacturing Execution System (MES) that can take advantage of end-to-end visibility, such as planning, production optimization, and supplier management.
- Asset management : This includes production-asset monitoring, as well as tracking and monitoring parameters areas, such as quality, performance, potential damage or breakdowns, bottlenecks, and so on.
- Field service organizations : These are an important driver of growth and, obviously, margin. It's clear that having a hyper-connected, hyper-aware, digitized, and IoT-enabled manufacturing ecosystem marks a company out.
- Remote monitoring and operation : This optimizes flow, eliminates waste, and avoids unnecessary work in the process inventory to save

energy and costs.

- Condition-based maintenance : This is important to optimize machine availability, minimize interruption, and increase throughput.
- AI : AI can be used to monitor the quality and the makeup of services and enhance the outcome of this data.

Ultimately, all of these use cases highlight the fact that data plays a key role. In the next few chapters, we will see how the data that comes from sensors and other industrial equipment is gathered and how big that data can be. Manufacturers who use this data can bridge the gaps between the planning, design, supply chain, and customers of a particular product. In addition, thanks to this strong integration, shared data and information islands of automation can easily be linked together.

IoT and IIoT – similarities and differences

There are many similarities between the IoT and the IIoT. The IIoT, however, is strictly related to industry and so it has some specific features, as highlighted in the following list:

- Cybersecurity is a critical topic for any digital solution, but its implementation in the industrial world requires special attention. This is because the OT systems and devices in the industry have a much longer life cycle and are often based on legacy chips, processors, and operating systems that are not designed to be connected over the internet. This means they live in an isolated LAN, protected from the external world by a firewall.
- It is critical to ensure that industrial digital devices stay running; any temporary disruption can imply a large economic loss.
- IIoT solutions must co-exist in an environment with a significant amount of legacy operation technologies. They must also co-exist with different devices acting as data sources, including SCADA, PLCs, DCS, various protocols and datasets, and back-office Enterprise Resource Planning (ERP) systems.
- Industrial networks are specialized and deterministic networks, supporting tens of thousands of controllers, robots, and machinery.
- IIoT solutions deployed into these networks must scale tens of thousands of sensors, devices, and controllers seamlessly.
- Physical objects in the industrial world are more complex and have a wider range of typologies when compared to the consumer world.
- In the industrial world, robustness, resilience, and availability are key factors. Usability and user experience, however, are not as relevant as they are in the consumer world.

- Industrial and OT systems, from programmable logic controllers to machining equipment, are frequently reprogrammed and reconfigured to support new processes.
- IIoT solutions must support and provide the same flexibility and adaptability to support operations.
- Intellectual Property (IP) is a sensitive and important topic in the industrial world. Consider, for example, the design of a new machine, engine, or food or drink recipe. The IP is often what differentiates a company in the market, and this cannot be lost or violated, since it is often managed by the company as a trade secret rather than covered through a patent.

IoT analytics, AI, and AIoT

With 15 billion industrial IoT devices deployed in 2023 and 29 billion expected for 2030, the volume of data generated is likely to reach 100 zettabytes (10^{21} bytes) per year.

A single jet engine produces about a terabyte of data in five hours. Given these assumptions, we need a fast and efficient way to analyze data through data analytics. In the last five years, big data technologies have been improved to scale computational capabilities. Analytics can analyze large datasets in order to discover value and hidden data and gain valuable information. The applications of these analytics are as follows:

- Diagnostic : Understanding the cause of a fault or issue
- Maintenance : Predicting and adjusting maintenance intervals to optimize scheduling
- Efficiency : Improving the performance of the production or the utilization of resources
- Prognostic : Providing insight to avoid faults or to maintain efficiency
- Optimization : Optimizing resource consumption or compliance with local government regulation
- Logistic and supply chain : Monitoring and optimizing the movement of goods and services from manufacturing to consumer via various means of optimized transportation

In the IoT, from a technical point of view, we can identify two broad categories of analytics:

- Physics-based : Based on mathematical formulas or knowledge expertise
- Data-driven (AI and machine learning) : The model is built using past data
- Hybrid Model : Physics-based and data-driven analytics can be

combined to build a reliable hybrid model

The introduction of deep learning (a branch of machine learning) in the contexts of image and text processing has brought a lot of attention to data-driven technologies.

Moreover, new generative AI technologies have brought further attention to AI.

"Artificial intelligence is nothing without data;
the IoT is nothing but data" – Antonio Capasso,
in the first edition of this book

Expanding the application of deep learning and in general AI to the IIoT is called AIoT . *AIoT is the application of AI to IoT infrastructure to reach more powerful insights and IoT operations.*

According to *Artificial Intelligence of Things: A Survey* (https://dl.acm.org/doi/10.1145/3690639),

> *"The integration of the Internet of Things (IoT) and modern Artificial Intelligence (AI) has given rise to a new paradigm known as the Artificial Intelligence of Things (AIoT)."*

However, there are two drawbacks:

- The (semi-justified) fear of companies of the potential pitfalls of AI
- The fact that traditional companies do not always trust the outcomes of AI

Resolving both of these issues will ensure that an abundance of caution is built into machine learning models used in industrial applications. We need to not only create better algorithms but also make sure that people with domain expertise understand machine learning suggestions. We also need to build systems that take in feedback and are aware of the end user and the effects of a good or bad response.

Industry environments and scenarios covered by IIoT

The industrial world is a very large category. It includes manufacturing, but also many other sectors, such as power and energy, renewable energies, health care, and so on. Within manufacturing itself, there is a large variety of sectors, including the automotive industry, chemicals, food and drink, and pharmaceuticals. Production is also only one phase of the product life cycle. Besides this, we have design, provision, delivery (with its own supply chain), and the aftermarket phase.

In this book, we will focus on the manufacturing environment by considering factory processes and strictly tailoring our analysis to the data. We will look at how data is produced, stored, processed, enriched, and exchanged between different OT systems inside industrial plants, and also at how it can be gathered, transferred, stored, and processed in the cloud. We will consider a scenario in which we have a specialized device, the edge device, which is responsible for collecting the data from the OT systems of the factory and transferring it to the cloud on a very large scale. We will also cover scenarios in which each edge device gathers and manages thousands of signals coming from sensors with a sampling rate starting from 1 Hz. The analysis and proposed solutions of these scenarios are also applicable to less complex cases in which there is no factory and/or fewer signals to manage. For example, consider a wind turbine, where you need to monitor a piece of industrial equipment. In this book, we will not cover scenarios in which there are too few signals to be collected for each data source to justify the need for an edge device.

Summary

In this chapter, we have analyzed the origin of the IoT and looked at how it came about through a combined set of technologies. We then learned about the key technologies that underlie the IoT, going into its use cases and business models. We defined the IoT as a technological layer that creates a digital twin of a physical object in the cloud, making the object more intelligent due to the interaction of its digital twin with other digital images living in the cloud. We also identified four levels to define the smartness of a product or object. We then looked at how the IoT can be applied to the industrial world, thereby beginning the Fourth Industrial Revolution and Industry 4.0. We looked at the key transformation elements that mark out the IIoT. We also highlighted some of the main use cases of the IIoT and the main differences between the IoT and the IIoT. We then listed and explored the different types of analytics that apply to industrial data. Finally, we clarified and defined the industrial scenarios that will be covered in the rest of the book.

In the following chapter, we are going to look at how a factory is structured and organized from an OT perspective. We will consider who produces, processes, and enriches the data. We will also explore some key concepts, including deterministic systems, real-time, closed-loop, sensor, fieldbus, PLCs, CNC, RTU, SCADA, HISTORIANS, MES, and ERP.

2

Understanding the Industrial Process and Devices

In this chapter, you will learn about how industrial data is generated, gathered, and transferred to the cloud. We will look at continuous or discrete processes and how they work, becoming familiar with the model of Computer-Integrated Manufacturing (CIM) from its origin in the factories of the 1980s to the current day. You will learn about industrial equipment, networks, and protocols, and come to understand terms such as Distributed Control System (DCS), Programmable Logic Controllers (PLCs), Supervisory Control and Data Acquisition (SCADA), Historian , Manufacturing Execution System (MES), Enterprise Resources Planning (ERP), and Fieldbus . We will also look at how the industrial world interacts with the cloud, as well as the devices and protocols that allow this to happen. Related to this, we will learn terms including OPC-Proxy, store and forward, edge, and IoT gateway. All the concepts that are sketched in this chapter will be further explained and analyzed over the next few chapters. We will provide a high-level, but all-encompassing, vision of the IIoT from a data perspective. We will look at the path of industrial signals, from their generation by the sensors to their processing in the cloud.

In this chapter, we will cover the following topics:

- The industrial processes
- The CIM pyramid
- The IIoT data flow

Technical requirements

In this chapter, we will present concepts related to industrial processes and equipment. Basic knowledge of analogical signal processing and analogical/digital and digital/analogical conversion is required. You will also need to be aware of elements of control theory, as well as LAN and WAN networking.

The industrial process

An industrial process can be defined as a set of operations that transforms, with a predetermined objective, the properties of one or more materials, types of energy, or pieces of information. A typical example of an industrial process might be the production of products and goods through a continuous or discrete manufacturing process, or the production of electricity, including its transportation and distribution. The industrial process also includes the collection, elaboration, and sharing of information along all production phases and steps. The industrial process is represented in the following diagram:

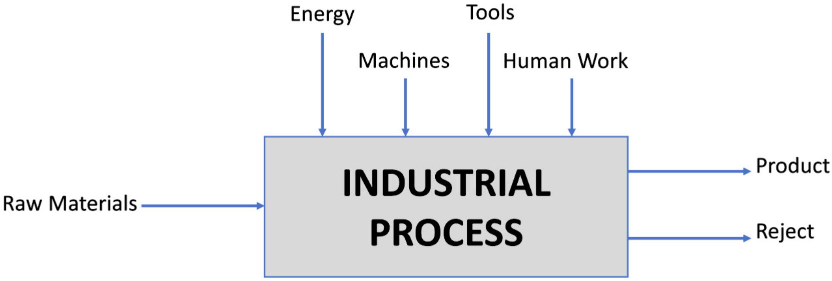

Energy
Tools
Machines
Human Work
Raw Materials
INDUSTRIAL PROCESS
Product
Reject

Figure 2.1 – Industrial process

This transformation from raw materials to a product requires the following:

- Energy
- Machines
- Tools
- Human work

The industrial process is a sequential process. It can be split into a further set of sequential production steps, transforming the raw materials into the desired state along the way, as shown in the following diagram:

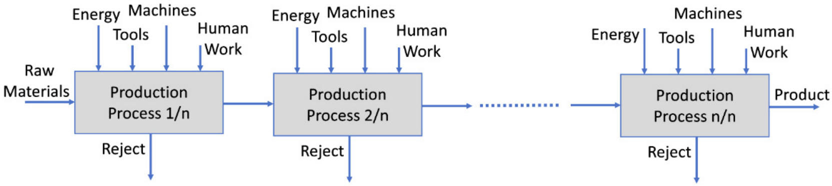

Energy
Machines
Tools
Human
Work
Raw
Materials
Production
Process 1/n
Reject
Energy
Machines
Tools
Human
Work
Production
Process 2/n
Reject
Machines
Energy
Tools
Human
Work
Production
Process n/n
Product
Reject

Figure 2.2 – Steps in the industrial process

Each production process is made up of a sequence of simple operations:

1. Making : This involves changing the material properties by means of energy.
2. Assembly : This involves combining one or more parts to make a new entity.
3. Transport and storage : Moving and storing parts, unfinished products, and products happens in this operation.
4. Testing : Checking the product to verify its functionality and capabilities against the design or the requirements is the key aspect of this operation.
5. Coordination and control : This involves coordinating and controlling the different operations and steps along the production process.

Automation in the industrial process

Factory automation can be defined as a discipline that studies the methods and technologies that allow the control of flows of energy, materials, and information needed for the realization of production processes.

The importance of automation in a modern production process derives from a multiplicity of factors, not just economic ones, among which are the following:

- The improvement of the quality of the products
- The opportunity to use the same production system for different products in a concept known as flexibility of the plant
- Shorter production times
- The opportunity to reduce the number of incoming and outgoing warehouses
- The drastic reduction of processing waste
- The lowered cost of production
- The need to comply with laws or regulations
- The opportunity to reduce the environmental impact and save energy
- The improvement of the competitiveness of the company in an automated system.

We can identify the physical processes and the control system, as shown in the following diagram:

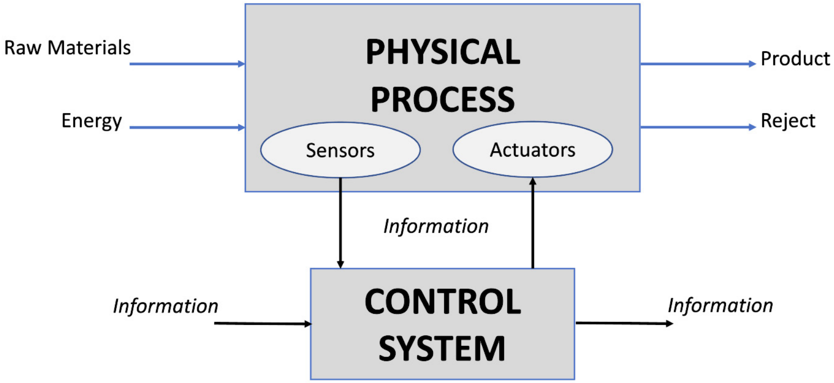

Raw Materials
Energy
PHYSICAL PROCESS
Sensors
Actuators
Product
Reject
Information
Information
CONTROL SYSTEM
Information

Figure 2.3 – Elements of an automated industrial system

Physical processes can be defined as the sum of the operations that act on entities belonging to the physical world and which change some of their characteristics. Operations that fit this definition include material or part movements, mechanical processing, or chemical reactions. These physical processes can be considered objects of automation. Pure and simple information, on the other hand, does not make changes to the real world, and so cannot be considered a physical process.

A physical process receives raw materials and energy as inputs. It also receives information, which can be in the form of electric voltage, current values, or fluid pressure, or which can be coded in sequences of binary values. It produces output materials, such as finished products and waste, and sends information. The noises coming from the environment that act on the process can also be considered as inputs to the process itself.

The outgoing information is provided by appropriate devices made by the following:

- Sensor : Transforms the variable to be measured into the type necessary for measurement
- Transducer : Accepts information in the form of a physical or chemical variable, and converts it into a magnitude of a different nature—typically electric

Very often, sensors and transducers coincide in the same physical component. We generally call a device a sensor (or a transducer) if it measures a magnitude and gives an output as a signal, typically an electrical one. The incoming information is used by the actuators to set the value of the control variables for the process. Usually, the real actuator is built by a pre-actuator, which processes the information to convert it into a power signal. Sensors, actuators, and pre-actuators carry out part of the physical process and act as interfaces to the control system.

A control system receives information on the status of the process from the sensors and processes them according to the specified algorithms. It then sends information related to actions that provide the desired control of the physical process to the actuator. The control system also receives information from one or more external entities, such as human operators or other control systems that are hierarchically higher. It is also able to provide information about its own status and the controlled process to the external entities.

Control and measurement systems

A more rigorous definition of what a process control and data acquisition system is has been provided by the International Electrotechnical Commission (IEC)-61131 standard. According to this standard, a control and measurement system of an industrial process can be described as a set of interconnected devices communicating with each other by means of one or more communication networks, as shown in the following diagram:

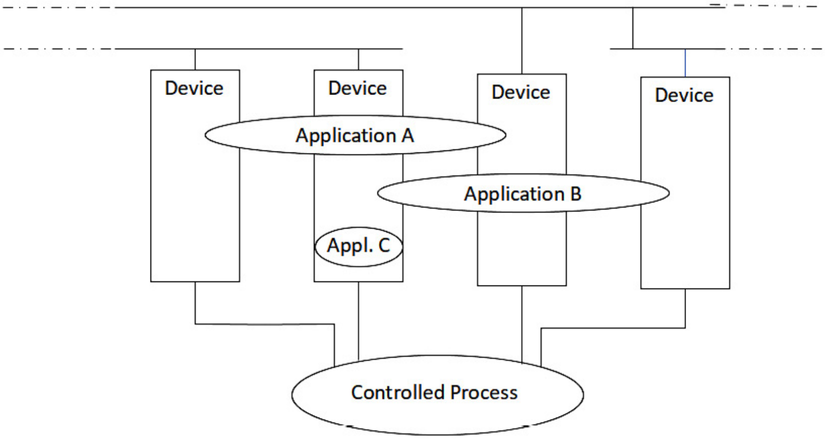

Device
Device
Device
Device
Application A
Application B
Appl. C
Controlled Process

Figure 2.4 – The control and measurement system model

The communication networks might be also structured hierarchically:

- Functionality performed by a system can be modeled as an application that may reside in a single device or be distributed among different devices. For instance, an application may consist of a control loop, in which the acquisition of the measures comes from one device, the control algorithm from another, and the commands are sent by yet another device.
- A device is defined as an independent physical entity that is able to implement one or more functionalities. The boundaries of a device are the interfaces made by the hardware and software components, which allow it to communicate with the outside world. A device must contain at least one resource and one interface for the process or communication network, as shown in the following diagram:

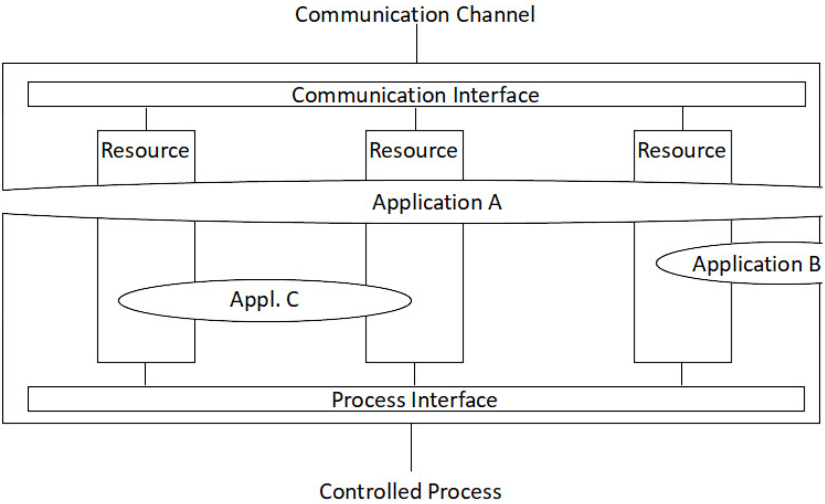

Communication Channel
Communication Interface
Resource
Resource
Resource
Application A
Application B
Appl. C
Process Interface
Controlled Process

Figure 2.5 – The device model

- A resource can be considered a logical breakdown of the software or hardware structure of a device that autonomously controls its operations. It can be created, configured, parameterized, started, or deleted without conditioning other resources in the same device. A resource receives data and events from the process and/or from the communication network, processes them, and returns data and/or events as requested by the application using it. In a resource, the following things must be present:

  - One or more local applications, or parts of distributed applications, which process the internal data and events
  - Functions linking internal data and events to and from the process and/or the communication networks
  - A planning function for coordinating the activities

- Data is representations of facts in a format suitable for communication and elaboration by the resource. An example of data is a binary representation of a measurement.
- Events are occurrences of specific conditions, such as the reaching of a definite temperature.
- A process interface links the resources contained in the device with the physical process by communicating with sensors and actuators. The information exchanged with the physical process is sent to the resources as process data or events related to the process.
- A communication interface links the resources with those of other devices to exchange information through a communication network. Through this interface, data or events are sent to the resource, and additional services such as programming support, system configuration, or diagnostic features are provided.
- An application specifies the operations that must be performed on the

data as a consequence of the events. It can be distributed between many resources in the same or different devices.

In the modern control and measurement systems for industrial processes, PLCs, DCS, and usual computers are used for this purpose. Generally speaking, PLCs and DCSs are connected directly to the sensors and actuators, while a usual computer exchanges information with other devices such as PLCs or other computers through communication networks.

Types of industrial processes

Industrial processes can be classified by whether their input and output are continuous or discrete. They can also be categorized by whether they belong to the process industry or the manufacturing industry, as shown in the following diagram:

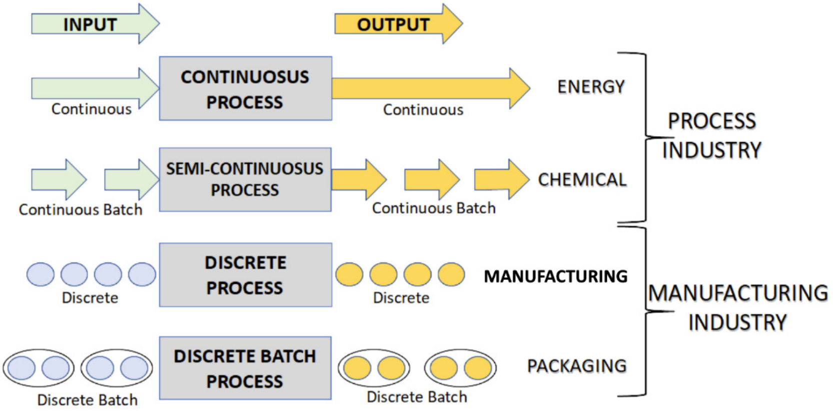

INPUT
OUTPUT
CONTINUOSUS PROCESS
Continuous
Continuous
ENERGY
SEMI-CONTINUOSUS PROCESS
Continuous Batch
Continuous Batch
CHEMICAL
PROCESS INDUSTRY
DISCRETE PROCESS
Discrete
Discrete
MANUFACTURING
MANUFACTURING INDUSTRY
DISCRETE BATCH PROCESS
Discrete Batch
Discrete Batch
PACKAGING

Figure 2.6 – Industrial process classification

A more precise classification splits each of these two main classes depending on the countability of their input and output. These classes are continuous, batch, semi-continuous, and discrete, all of which are described in the following sections.

Continuous processes

Continuous processes involve the continuous mass transformations of energy and momentum on a flow of material. In a continuous process in normal operation, the goal is to obtain a product of uniform quality over time, regardless of how long the process has remained active. A process may last days or even weeks, and may even only be turned off for verification operations, cleaning, and maintenance (either scheduled or unplanned due to failure). Continuous processes, which are also called industry or process controls, are as follows:

- Energy production
- Distribution of energy, water, and gas
- Crude oil and gas extraction
- Rolling plants
- Hydraulic systems for the collection and distribution of liquids or gases
- Ovens or driers
- Production of glass, cement, paper, expanded clay, and so on

Batch processes

Batch processes involve finite quantities of the final product, which are obtained from finite quantities of raw materials and then processed according to an ordered set of activities in a finite time interval. Batch processes can be applied to the following:

- Products that are not processed in a fixed, predetermined quantity

- The quantity of the final product, which depends on the recipe
- Production processes that are interrupted after a lot of processing has been done, before resuming again; in this case, there are no parts that can be easily separated and identified
- Equipment used for different products, but with an intermediate cleaning phase needed

Typical examples of batch processes include the following:

- Products made according to different formulas, but in small or medium quantities, such as pharmaceuticals, detergents, or plastics
- Expensive products with a restricted market or reduced life cycle, such as perfumes
- Products that require maturation, fermentation, or mixing times in regulated conditions, such as wine, beer, other alcoholic beverages, or oil

Semi-continuous processes

Semi-continuous processes have some characteristics in common with both continuous and batch processes. Typical examples of semi-continuous processes are the following:

- Filtering or cleaning of gases or liquids

- Dehumidification of air
- Water treatment
- Separation of components in a continuous flow

Discrete processes

Discrete processes are characterized by processing cycles based on single parts or individual units of a product. In a discrete process, both the raw materials and the final product are countable. Typical examples of batch processes are as follows:

- Mechanical machining (such as turning, milling, drilling, or welding), assembly, handling, welding, painting, or quality control
- Transport of products, pallets, tools, belts, rollers, conveyor chains, automatic guided trolleys or Autonomous Guided Vehicles (AGVs) on fixed tracks, magnetic tracks, or bridge carriages (over high weights or long distances)
- Buffers used to store up products (either raw, semi-finished, or finished) or tools for managing ongoing queues to an operating machine and absorbing different processing speeds

The CIM pyramid

The production support system is a set of activities for managing the flow of information related to production. It includes the following aspects:

- Business : These are activities that strictly face the customer and are related to the start and end of the whole process. These include order management, marketing, sales, and the budget.
- Design : These are activities related to designing the product according to the customer's needs, expectations, and requirements.
- Planning : These are activities based on planning the business and design functions of a product. They involve the working sequence, timing, storage, and supplies.
- Control : These are activities related to the management and supervision of the production process. It includes controlling the production flow and checking the quality of the production processes and products.

We can sketch these activities as a loop around the production activities managing the related flow of information, as shown in the following diagram:

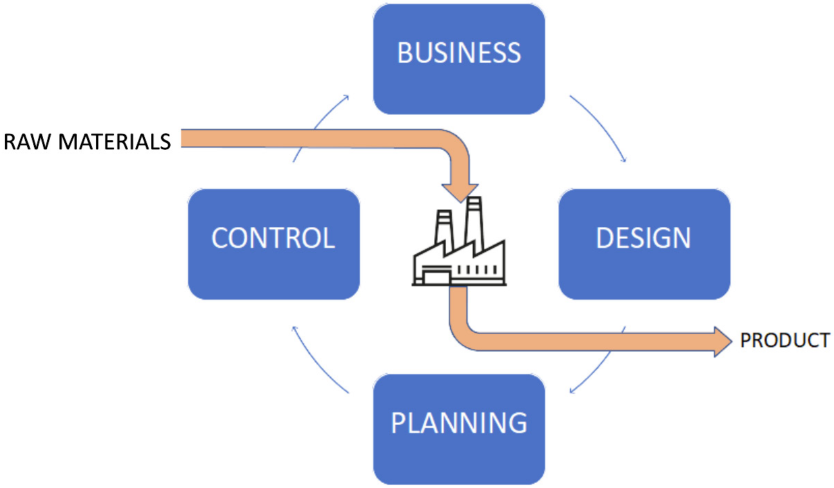


Figure 2.7 – Production support systems

For all of these, there are specific software applications for automating and coordinating the different activities. These include the following:

- ERP : For managing tasks such as logistics, production management, accounting, personnel, purchases, warehouses, project and sales management, and plant and distribution management
- Computer-Aided Design (CAD) and Computer-Aided Engineering (CAE): Software tools for designing, testing, and validating the product against the specification or requirement
- Computer-Aided Process Planning (CAPP) and MES : Software applications for automating and optimizing the planning process

This type of approach, however, might lead to the creation of automation islands that are not integrated with each other. A huge improvement can be achieved by integrating all subsystems at the company level. This can be done as follows:

- By implementing a structured design of the whole information technology stack
- By managing the flow of information between the various devices at different levels
- By coordinating factors involved in the production cycle, including the human factor, as shown in the following diagram:

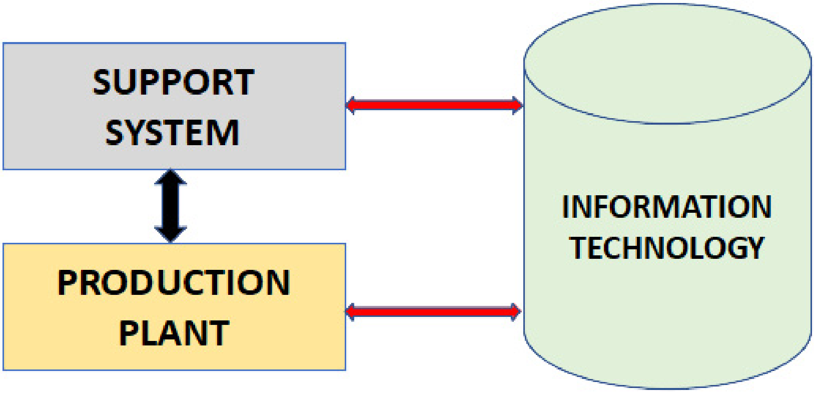


Figure 2.8 – The CIM logical schema

However, in all production processes, there is a trade-off between the variety and the quantity of the production in terms of the number of parts or the output. Typically, the higher the quantity of the production, the less variety there can be in the production.

This trade-off threshold can be then shifted up through a very flexible production system that uses highly configurable machinery. This machinery can be easily set up for different uses and allows for a strict interaction and integration between the information systems of the production process and the support activities, as depicted in the following diagram:

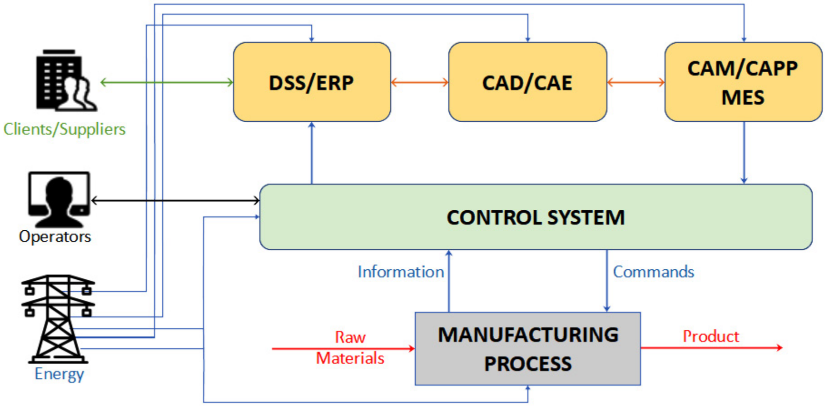

DSS/ERP
CAD/CAE
CAM/CAPP
MES
Clients/Suppliers
Operators
CONTROL SYSTEM
Information
Commands
Energy
Raw
Materials
MANUFACTURING
PROCESS
Product

Figure 2.9 – The CIM model

CIM is a logical model for production systems that was developed in the 1990s to integrate production processes, automation systems, and information technology systems at a company or enterprise level.

CIM should not be considered as a design technique for building automatic factories, but rather as a reference model for the implementation of industrial automation based on the collection, coordination, sharing, and transmission of data and information between the different systems and sub-systems by means of software applications and communication networks.

The CIM model is strongly hierarchical:

- Support activities are at a higher level than those of production. There is a hierarchy within those support activities.
- The business activities influence the design and planning of production.
- The production activities are also structured in a hierarchy. The automation of a production step is at a lower level than the automation of the whole machine, which is carried out by a further sequence of steps.

The CIM model is often depicted as a pyramid made up of six functional levels, as shown in the following diagram:

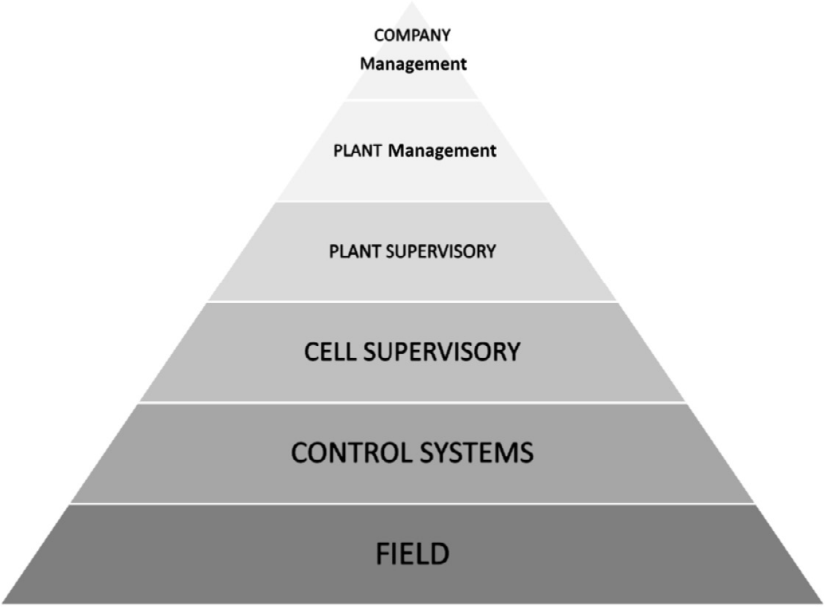

COMPANY
Management
PLANT Management
PLANT SUPERVISORY
CELL SUPERVISORY
CONTROL SYSTEMS
FIELD

Figure 2.10 – The CIM pyramid

- Level 1—field : This level includes all devices that interact directly with the process, such as sensors and actuators.
- Level 2—command and control systems : This level includes the devices that interact directly with the sensors and the actuators, such as PLCs, micro-controllers, Proportional-Integral-Derivative (PID) controllers, robot controllers, and Computer Numerical Controllers (CNCs).
- Level 3—cell supervisory : In a cell, a complete sub-process of production is executed through various devices and machines that must be coordinated with each other. The main functions of the control devices placed at this level are the following:

    - The receipt of the instructions from the upper level
    - The transformation of the lower-level devices into actions and commands
    - The collection of information from the lower levels to pass to the upper level
    - The arrangement of information for the human supervisor, who may eventually issue commands or set up set-points or thresholds

- Level 4—plant supervisory : At this level, the production database collects and stores the main parameters of the production process, and the coordination between the cells is carried out to implement the whole production process.
- Level 5—plant management : At this level, the aforementioned integration between the support systems takes place.
- Level 6—company management : Typically, a company handles several plants. Therefore, at this level, the information is collected from the lower levels to feed the decision support systems that help managers plan flows of materials and finance necessary for the maintenance, improvement, and optimization of the production process.

The pyramidal shape used to present the CIM levels is suitable for a hierarchical organization in which the following occurs:

- Each level communicates directly with the upper one, from which it receives commands and to which it sends information

- Each level communicates directly with the lower one to send commands and receive information
- Each level sends, to the upper level, less information at a lower frequency than that received from the lower level

This strict and mutual interaction between the different systems and sub-systems involved in the production cycle has a number of benefits. Among the most important of these are the following:

- More efficient usage of resources through the planning of production processes
- Greater flexibility in production, as the system can be promptly and easily adjusted to conform to a new process
- A reduced processing length
- An improvement of the product design due to digitization, given the need to provide unambiguous information to digital machines and systems
- The identification, storage, and re-use of information related to the product
- An improvement in the quality of the product due to more stringent validation checks
- The reduction of stocks of raw materials and warehouses used to store finished products and any processing waste

It is important to remember that the CIM pyramidal structure is a logical representation of the factory automation world. This means that the canonical six levels can be coalesced into a simpler structure with fewer levels. From an operational perspective, for instance, the CIM pyramid is often represented in five levels:

- Field level : Sensors, actuators, and hardware
- Control level : Control systems, such as PLCs and DCSs
- Supervisory level : SCADA systems, time series databases, recipe management, production reporting, and alarm management
- Planning level : MESes, plant-wide operations, asset management, and maintenance
- Management level : ERP systems

In IT companies that provide automation services to factories, the CIM pyramid is typically simplified to a three-level structure:

- Automation control : All equipment for the automation and control
- Supervisory : Devices and applications for the supervision of the production process and the maintenance of equipment
- Production management : Applications for the planning and management of the production

Both coalesced representations of the CIM pyramid are shown in the following diagram:

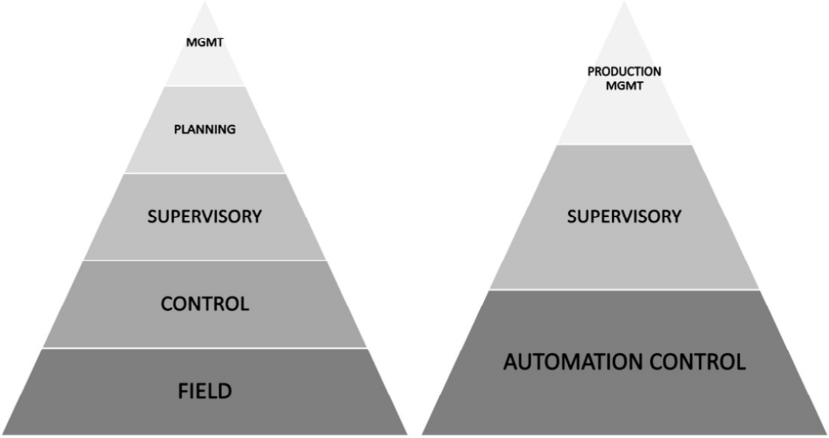

MGMT
PLANNING
SUPERVISORY
CONTROL
FIELD
PRODUCTION MGMT
SUPERVISORY
AUTOMATION CONTROL

Figure 2.11 – Two simplified versions of the CIM pyramid

CIM pyramid architecture – devices and networks

The following simplified schema represents a typical automation architecture with reference to the CIM pyramid. Each level, shown in the center, has been linked on the left-hand side to the related communication networks, and to the devices or applications involved on the right-hand side:

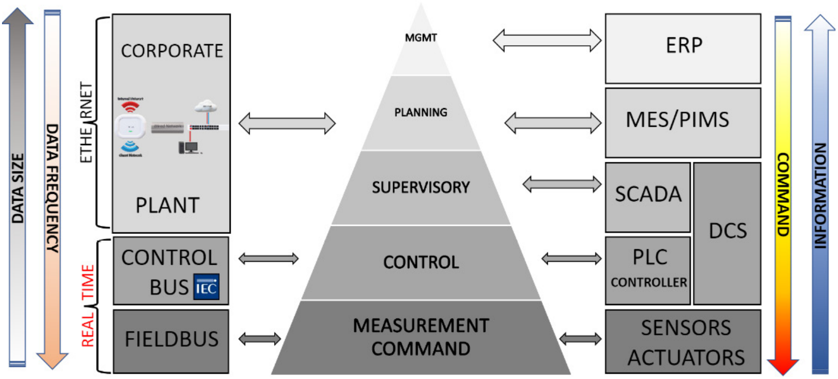

DATA SIZE
DATA FREQUENCY
ETHERNET
REAL TIME
CORPORATE
PLANT
CONTROL BUS
IEC
FIELDBUS
MGMT
PLANNING
SUPERVISORY
CONTROL
MEASUREMENT COMMAND
ERP
MES/PIMS
SCADA
DCS
PLC CONTROLLER
SENSORS ACTUATORS
COMMAND
INFORMATION

Figure 2.12 – CIM devices and networks

The control devices and the industrial networks will be analyzed and discussed more thoroughly in *Chapter 3* . In this section, we will provide a description of each of these industrial systems. This will introduce the next section, which will present the flow of industrial data from the factory to the cloud.

A description of the devices and applications related to the CIM pyramid is given in the following section.

Level 1 – sensors, transducers, and actuators

According to the IEC, the sensor is defined as the following:

"The primary element of a measurement chain, which converts the input variable in a signal suitable to the measurement."

The transducer is the following:

> *"The device that accepts information in the form of a physical variable (its input variable) and converts it to an output variable of the same or of different magnitude, according to a defined physical law."*

Since the sensor and the transducer are often physically within the same component, the two terms are often used as synonyms. The actuator is a transducer that transforms a command signal into a physical action in the process. Basically, its function is complementary to that of the sensor. It receives a signal as an input in the physical domain of the control device and sends energy as an output in the physical domain of the command variable. They are often made by means of multiple transducers.

Level 2 – RTU, embedded controllers, CNCs, PLCs, and DCSs

Level 2 encompasses all devices that directly interface with sensors and actuators, executing control and safety functions for individual equipment, work cells, and entire production lines. They are the following:

- A Remote Terminal Unit (RTU) is an electronic device that is controlled by a microprocessor. It acts as an interface for sensors, actuators, and intelligent devices by transmitting data and receiving command messages from a master system. It is basically used to centralize the input and output to and from the sensors and actuators by reducing the complexity of their cabling and wiring.
- An embedded controller is generally a single chip or board that includes all the necessary components to carry out the required control tasks. They are usually designed for a specific application and are built on top of specific or custom hardware.
- Computer Numerical Controls (CNCs) are machine tools that are controlled by an electronic device integrated into the machine. Movements and functions of CNC machines are predefined and set through specific software. They are used for performing high-precision machining that requires long processing times without any interaction with the external environment.
- A PLC is an industrial controller that is dedicated to the control of industrial processes. The PLC executes a program in a cyclical fashion, processing the signals coming as input from the sensors and sending the output values to the actuators to control the physical process. The reading of the inputs, their processing, and finally the writing of the outputs, takes place within a predefined maximum time called a scan cycle. This typically takes between 10 and 100 milliseconds.

- DCSs are typically used in continuous processes such as in refineries, energy production, or chemical plants. They integrate both the control function implemented in the PLC and the supervision of the SCADA system. While the PLC and the SCADA are two separate systems, each with their own variables and data structures, in the DCS, the control and the supervision of their processing tasks share the same variables and data structures.

Level 3 – SCADA, Historian

The SCADA system is not a specific technology like PLC or CNC, but instead refers to vertical and tailored applications that are based on computerized and centralized systems to control industrial machinery and processes remotely and in real time. SCADA is linked with a data acquisition system, which allows it to gather real-time information on the operating status of the machines. A SCADA system implements the following main functionalities:

- It receives data and information from the PLCs within the control system, which in turn gather input from sensors, RTUs, and other devices situated at the lower levels of the CIM pyramid.
- It processes the gathered data, storing the most relevant information.

- It detects process anomalies, triggering alarms that are escalated to the operators in the control room.
- It presents information to the operators in the control room through a Human Machine Interface (HMI).
- From the HMI synoptics, the operator can monitor the data of the overall plant under the supervision of the SCADA application. If needed, the operator can also react promptly through the sequences and operational logic implemented in the SCADA system by sending commands to the PLCs.

Level 4 – MES

The MES is a software system located between the ERP and the SCADA or PLC to manage the production process of a company efficiently. Its main function is to synchronize the business management and the production system by bridging the planning and control levels to optimize processes and resources.

The main features of an MES are as follows:

- Order management and production planning
- Management of the incoming raw materials and semi-finished products

- Asset management and monitoring
- Production tracking
- Maintenance management
- Quality checking

Level 5 – ERP

ERP includes the systems and software packages that organizations use to manage the daily activities of their business, such as accounting, procurement, project management, financial, warehouse, logistics, and production. ERP systems combine and define a set of business processes governing the exchange of information and data between the systems involved. ERP collects and shares transactional data coming from different departments of the organization, thus ensuring data integrity by acting as a single source of truth .

CIM networks

At the lower levels, we have a large amount of simple information that must be transferred within small and predefined deterministic time intervals. At the higher levels, the requirement is to transfer much more complex information within time intervals that do not need to be deterministic. So, in an integrated production system, different types of communication networks are required, each one specialized for the task to which it is dedicated. With regards to the CIM pyramid, we have the following networks:

- Level 1 : Fieldbus
- Level 2 : Control bus
- Levels 3, 4, and 5 : Corporate network

Field or fieldbus networks were introduced for interfacing control devices, sensors, and actuators, reducing the need for complex wiring. In the fieldbus, sensors and actuators are equipped with a minimum set of processing capabilities to ensure the transmission of information in a deterministic way.

The control network must ensure the communication between the devices is dedicated to the control and supervision of the plants. In this case, the information sent is not very complex, but the transmission must occur within predefined deterministic time intervals and at higher frequencies. Control and fieldbus networks are also often called real-time networks because of their determinism in the transmission of data and information.

The corporate network is the network channel located between the control, planning, and management systems. This connectivity layer must guarantee the processing of complex information but at low frequencies. For this reason, there is no need to have determinism on this network.

The IIoT data flow

In this section, we will introduce the whole flow of industrial data from the sensors to when data is processed in the cloud. We will look more closely at data acquisition, data sources, the related protocols, and the edge device to push the data into the cloud in *Chapter 3* and *Chapter 4* . We will then look at an overview of cloud architecture and the main components for the IIoT in *Chapter 7* , and beyond.

Here, when explaining the IIoT data flow, we have not mentioned cybersecurity. This is not because the topic is not pertinent or relevant. On the contrary, cybersecurity is always a hot topic, especially in an industry in which the availability and reliability of equipment is a must, and confidentiality and intellectual properties are increasingly important. In addition, opening systems and devices that have been isolated until now, and not designed to be linked to the internet, has put them under stress from the cyber-security perspective once their isolation is broken. For this reason, *Chapter 5* , will be dedicated to discussing cyber-security in Industrial Control Systems (ICSs) and Intellectual Property (IP) in the IIoT.

The following diagram shows the IIoT data flow from the sensors to the cloud:

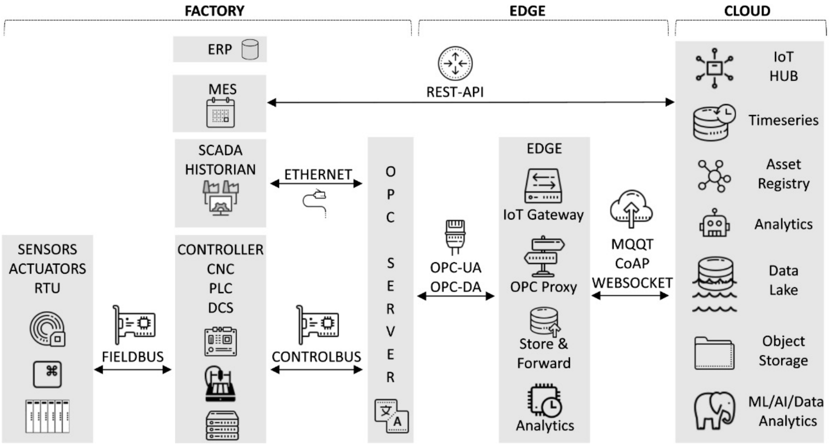

FACTORY
EDGE
CLOUD
ERP
MES
REST-API
SCADA
HISTORIAN
ETHERNET
OPC SERVER
EDGE
IoT Gateway
OPC Proxy
Store & Forward
Analytics
MQQT
CoAP
WEBSOCKET
OPC-UA
OPC-DA
SENSORS
ACTUATORS
RTU
FIELDBUS
CONTROLLER
CNC
PLC
DCS
CONTROLBUS
IoT HUB
Timeseries
Asset Registry
Analytics
Data Lake
Object Storage
ML/AI/Data Analytics

Figure 2.13 – The IIoT data flow

The Industrial IoT data flow in a factory

The best way to explain the IIoT data flow of industrial signals, from their generation at the field level in the factory to when they are processed in the cloud, is to start from the controllers. They perform the following steps cyclically and sequentially:

1. Read the measurements that arrive as input from the sensors.
2. Execute the control algorithms that are fed by the last reading of the inputs.
3. Write the outcomes of the control algorithms as outputs to the actuators.

This cycle is the same for all kinds of controllers (embedded controllers, PLCs, or DCSs, for example) but a cycle's duration varies according to the kind of controllers in use:

- Embedded controllers take a few milliseconds PLCs take between 10 and 100 milliseconds
- DCSs take between 200 and 1000 milliseconds

In any case, any controller, in every scan cycle, whatever its duration is, performs a reading of all the sensors and calculates the values for all actuators linked to it. For example, a PLC that controls a gas turbine can manage up to 3,000 readings coming from the sensors or as feedback commands from the actuators. Basically, this means that 3,000 values are read every 100 milliseconds, 30,000 every second, 1,800,000 every minute, 108,000,000 every hour, and 2,592,000,000 each day. This is just for one PLC controlling a single piece of equipment, even if that piece of equipment is quite complex, such as a gas turbine.

The values refer to analog signals, such as temperatures, pressures, or Boolean for two-status representation, such as whether a valve is open or closed, or whether a part in a conveyor is present or not. The sampling of these values comes as an input to the systems of the upper level, such as SCADA and Historian, as a representation of the related signals. In the industrial world, the digital representation of a signal is called a tag, and its samples are called time series. A time series is an ordered sequence of timestamps and values related to a signal in a particular time interval. SCADA systems and Historian gather the data from the controllers but with a higher scan time. Typically, they collect the time series using a scan cycle from 500 to 1,000 milliseconds in SCADA applications, and from one second to one minute or more in the Historian systems.

Collector compression applies a smoothing filter to incoming data by ignoring incremental changes in values that fall within a deadband centered around the last reported value. The collector reports any new value that falls outside the deadband to the Historian archive and then centers the deadband around the new value.

In addition, a smoothing filter is applied to the sample of an analog signal that is gathered by the SCADA and Historians by ignoring incremental changes in values that fall within a deadband, which is centered around the last reported value. Samples of the Boolean signals are typically collected when their state changes rather than at every scan time. In this way, the amount of data that SCADA and

Historian must deal with is much less compared to that managed by controllers, but it has a lower time resolution. The balance between the amount and the resolution of the data collected is a trade-off, depending mainly on the dynamic of the industrial process and the resources that are available. PLCs and DCSs can also provide other information to the upper levels besides the time series, such as the following:

- A memory buffer area temporarily stores the values of the internal variables of the controllers at its scan time in a sliding time window. The memory buffer area can be exported as a binary or text file on an event or a condition.
- Alarms triggered by measurements or calculations that exceed the configured high or low limits for that variable.

From SCADA, the most important information flows to the MES, and vice versa. At this level, the information provided by SCADA can be the total amount of a batch, the timestamp of its start or stop, quality samples, and so on. The information received from the MES might be the recipe to use, a cleaning cycle to be carried out, the kind of product or part to be produced, on which equipment it should be carried out, and so on.

From the data collection perspective, the data sources of the IIoT are the following:

- PLCs and DCSs at level 2 of the CIM pyramid
- SCADA systems at level 3 of the CIM pyramid
- Historian systems at levels 3 and 4 of the CIM pyramid

Sensors, RTUs, CNCs, single pieces of equipment, or stations that are driven by an embedded controller cannot act autonomously as a data source for the IIoT gateway. This is because they have too many connections to isolated networks that are managed by legacy protocols. Moreover, their data is mostly collected and made available by the PLCs and DCSs, so there is no need to connect directly to them.

PLCs and DCSs in the control network are managed by fieldbus protocols, such as Modbus, Profibus, and

ControlNet. Fieldbus protocols are specialized industrial network protocols that ensure determinism in the packet delivery to the devices that are configured as belonging to that network. PLCs and DCS manage all data coming from the sensors and devices of the lower levels according to the resolution of their scan time. Control networks are isolated networks that are protected by a firewall. They have a few well-controlled ports that are open to the upper levels by means of firewalls.

SCADA and Historian are in the corporate network and are based on Ethernet, so they are not in an isolated and specialized network and therefore don't need specific network cards or boards to communicate with the other systems. SCADA and Historian do not collect all the data that is made available by the PLCs to which they are connected. Instead, they collect the most important data, as well as that which has a lower frequency.

In any case, whatever the device that is acting as a data source is, we need to have a software layer to interact with the industrial data source through its own protocol for querying tags and time series and exposing them by means of a standard interface to the upper levels and external systems. This software layer, which acts as a translator between the controllers and the producers of the industrial data and the consumer systems, is the Open Platform Communications (OPC). The OPC is a standard that is used to abstract PLC or DCS-specific protocols (such as Modbus or Profibus) into a standardized interface. This allows SCADA and other systems to interface with a middle-man which converts generic OPC read or write requests into device-specific requests and vice versa.

Through the OPC, the PLC or the DCS can expose tags, time series, alarms, events, and Sequences of Events (SOEs) to any system that implements the OPC interfaces. In the next chapters, we will look at how the OPC standard has changed from its first specification in 1996, when it was called OLE for process control, to the current specification, which is now called OPC. It has overcome various challenges related to security, data modeling, and performance in manufacturing systems. Since their beginnings, OPC specifications and the related interfaces have enjoyed considerable success, becoming, over the course of a few years, the de facto standard to expose industrial application data and make it available to external systems.

SCADA and Historian were initially developed to be consumers of industrial data. By implementing an OPC client interface, they themselves became producers of industrial data, thus implementing an OPC server interface and playing both roles at the same time. The quality and performance of the OPC technology had various issues and limitations at the beginning. Even so, the adoption and spread of OPC in the industrial world was considerable. This was because of the significant need of all the actors involved, which includes vendors, end users, and integrators, to exchange data between applications belonging to the automated production system in a standardized and reusable way. OPC interfaces allow you to query for time series and data cyclically, both on changes and through subscriptions with dead bands. They have a scan time of typically one second. Cyclical data collection is also called polling or just polling, on-change data collection is often referred to as unsolicited, and the publish/subscribe mechanism is also known as subscription.

The OPC standards include two main specifications for OPC interfaces: OPC Classic and the OPC Unified Architecture (UA). These specifications have no compatibility with each other.

The edge device

The edge device is the device that enables the IIoT.

It is physically located in the factory and is linked, from one side, to the industrial data sources through the OPC server, and from the other side to the cloud through the IoT gateway. There is much debate about the location, role, and responsibilities of the edge device. With regard to the IIoT, it must have the following minimum capabilities:

- Implement OPC client interfaces (both OPC Classic and OPC UA) to gather the data from the industrial data sources, which include PLCs, DCSs, SCADA, and Historians
- Have OPC Proxy functionalities to manage indifferently the incoming data requests following both OPC Classic and OPC unified interfaces and addressing them accordingly
- Implement bi-directional communication to the cloud through VPNs, SSLs, or cellular lines through the most common internet application protocols such as Message Queuing Telemetry Transport (MQTT), Advanced Message Queuing Protocol (AMQP), and Hypertext Transfer Protocol Secure (HTTPS) to do the following:
    - Transfer the data to the cloud through the IoT hub
    - Receive commands and configurations from the IoT hub
    - Implement a store and forward mechanism to guarantee the data transfer along the communication channel in case of poor, intermittent, or overloaded connectivity

    - Expose centralized functionalities available from the cloud, including the following:
        - Setup and monitoring of the device
        - Data acquisition parameters and configuration deployment
        - Software patches and software upgrade deployment
    - Be available both as a physical and a virtual appliance
    - Be multi-platform (at least on Windows and Linux)
    - Be flexible and scalable to support the collection and transfer of anything from several hundred tags to dozens of thousands of tags

Additional capabilities of the edge device might include the following:

- Capturing, storing, and processing the buffer with high-frequency data (from 1 Hz to 20 kHz)
- Implementing drivers for the most common control bus protocols (PROFINET, EthernetIP, and Modbus)
- Implementing an analytics engine to detect anomalies and/or locally processing high-frequency data related to the transient or high dynamic phenomenon
- Implementing an Advanced Process Control (APC) to analyze the

process and optimize it autonomously, sending the suggested outputs to the controllers

- Implementing a Trusted Platform Module (TPM) to improve cybersecurity

The Industrial IoT data flow in the cloud

The most relevant components that manage the flow of industrial data in the cloud are the following:

- IoT hub : This is the dispatcher of data and the manager of devices. It checks security and dispatches data to the right data processors (storage, analytics, or queue). Normally, it is implemented with a multi-protocol gateway, such as AMQP, HTTPS, or MQTTS, and a message broker.
- Time Series Database (TSDB): This is the centralized database in which events and data points acquired from sensors are stored.
- Analytics : These work with data to extract anomalies, the machine's health, efficiency, or generic Key Performance Indicators (KPIs). Analytics can work either in stream mode or in micro-batch processing mode. Normally, we use simple, stream-based analytics to evaluate simple rules, and Machine Learning (ML) analytics or physics-based analytics for more complex analytics working in micro-batch mode.
- Asset registry : This supports additional (static) information, such as

the model of the machine being monitored and the operational attributes. This might include the fuel used, the process steps followed, or the machine's status.

- Data lake : This is normally used to support raw data, such as images or log files. Sometimes, it is used to offer storage support for events and measures (time-series). We normally store the outcome of the analytics in the data lake.
- Object storage : This stores additional information for large files or document-based data. Object storage can be implemented using the data lake.
- AI/ML/data analytics : These are not necessarily within the scope of the IoT, but sometimes we need to run data analytics over the entire fleet. Alternatively, we might be using a huge amount of data to carry out business analysis.

Summary

In this chapter, we looked at what an industrial process and an automated system are. We identified the physical process, the control systems, and their interactions. The physical process is a combination of operations that act on entities that belong to the physical world and change some of their characteristics. The control system receives the information on the status of the process, elaborates it, and performs the required actions on the physical process. A more rigorous definition of process control and data acquisition systems was also provided. We identified and defined the different entities involved, which included devices, resources, data, events, and the interface.

Following this, we learned about the different kinds of industrial processes there are, distinguishing between continuous, semi-continuous, discrete, and batch processes. We also looked at which activities support the production cycle, including business, design, planning, and control. After that, we introduced the CIM model, which is a hierarchical structure that integrates production processes, automation systems, and information technology systems at a company or enterprise level. We analyzed, in detail, the hierarchical structures of the CIM pyramid, acknowledging its levels and their aggregation from different perspectives. We then learned what the main industrial devices and networks are at each level of the CIM pyramid. From this, we gained an understanding of which devices act as data sources in the IIoT data flow.

Following on from this, we analyzed the data flow of industrial signals, from their generation by the sensors, to when they are processed in the cloud. Data acquisition, data sources, the related protocols, and the edge device for pushing the data into the cloud will be analyzed and discussed in *Chapters 3* and *4* . In *Chapter 5* , ICS and IP in the industrial IIoT will both be discussed in full.

3

Industrial Data Flow and Devices

In this chapter, we'll take a close look at the equipment, devices, protocols, and software layers that manage industrial data along its path in the factory. We'll start from the sensors and track the path of data until we reach the edge, which is the external boundary of the I-IoT data flow inside the factory. We'll look at how industrial signals are generated and how they are managed, exchanged, and enriched by passing through the different levels of the CIM pyramid.

We'll also consider what the main OT devices involved in each level are. Furthermore, we will learn about analog to digital conversion and vice versa, sensors and actuators, Remote Terminal Units (RTUs), embedded controllers, Programmable Logic Controllers (PLCs), Distributed Control Systems (DCSs), Supervisory Control and Data Acquisition (SCADA), Historian, industrial protocols, and fieldbus. We would require a specific chapter for each system in order to explain them exhaustively, so instead, we will focus on analyzing them in this chapter from the point of view of the IIoT data flow. We'll briefly highlight their main mechanisms, but we'll explain in detail how they are important and relevant from the perspective of data acquisition and collection. Acquiring industrial data from sensors and controllers is one of the most critical processes in IIoT and comes with its own specific challenges and constraints.

In this chapter, we will cover the following topics:

- The IIoT data flow in the factory
- Measurements and the actuator chain
- Controllers

- Industrial protocols
- SCADA
- Historian
- ERP and MES

Technical requirements

You will need to have read the previous chapters of this book to understand the concepts presented in this chapter. Knowledge of the following topics is also necessary:

- Networking and the related ISO/OSI model
- The basics of electronic circuits
- Key elements related to computer architecture

The IIoT data flow in the factory

The following diagram depicts the main pieces of equipment, OT devices, networks, and protocols involved in the IIoT data flow. These are arranged according to their hierarchical structure in the CIM pyramid, as we have already seen in previous chapters:

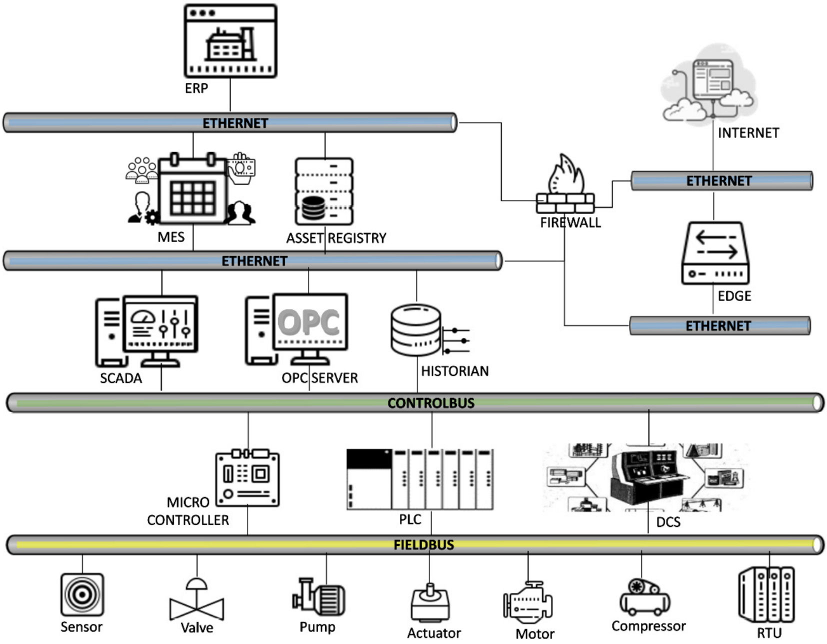

ERP
ETHERNET
INTERNET
MES
ASSET REGISTRY
FIREWALL
ETHERNET
ETHERNET
EDGE
ETHERNET
OPC
SCADA
OPC SERVER
HISTORIAN
CONTROLBUS
MICRO CONTROLLER
PLC
DCS
FIELDBUS
Sensor
Valve
Pump
Actuator
Motor
Compressor
RTU

Figure 3.1 – IIoT devices and protocols in the factory

We will emphasize here that this logical schema is not a complete picture of all of the equipment, devices, networks, and protocols that we have in an industrial plant—nor is it the only possible representation of their interactions and interconnections. As we mentioned in previous chapters, there is no such precise and clear-cut separation of functionalities and devices between CIM levels in the real industrial world. Some devices placed in one particular level can be deployed to an upper or lower level with an enlarged or restricted scope. This is even more true nowadays, with the progress of this technology providing additional capabilities and more computational resources to devices and allowing them to enlarge their initial scope of functionality. Let's take a look at a few examples to understand this better:

- Sensors and actuators : Since these exchange very little information, they are traditionally placed in level 1. Over the past few years, however, smart sensors have been introduced with processing and computing capabilities to exchange and provide much more information.
- PLCs : These are generally placed in level 2. However, we can also have powerful PLCs in level 3, acting as control coordinators and data collectors. We can even have micro PLCs between levels 1 and 2 to control a working cell or to drive a robot or a complex piece of equipment such as a rotating machine.
- SCADA systems : These are located in level 3. We can also have a local

Human Machine Interface (HMI) in level 2 to supervise working cells that cooperate to carry out a specific task locally. SCADA systems have been evolving over the last few years, covering tasks such as tracking, maintenance, and production planning. These are traditionally covered by the MES system in level 4.

- Historians : These are placed in level 3, but we often have a time window snapshot that pushes the time series to a larger replica database that is located in level 4.

Cybersecurity is not covered in this schema, except for a generic firewall that is placed between the factory and the edge that is exposed to the internet. We will learn how to secure the IIoT data flow in *Chapter 5* .

The preceding diagram focuses on the IIoT data flow. It shows the devices that act as data sources and highlights the network channels and protocols that allow the data and information to pass along the devices, as well as the layers of the CIM pyramid, until they reach the edge and are transferred to the cloud. Let's start by looking at where the digital signals are generated.

Measurements and the actuator chain

The devices that measure and act on the process variables are of fundamental importance in implementing a control system. Their values are used by controllers, together with the reference values that are inputted, to generate the control signals. This is outlined in *Figure 3 .2* , where we have the following:

- Set point (SP): This is the reference and the desired value for a specific process variable
- Controlled value (CV): This is the calculated value for the control variable to fit the reference value
- Process value (PV): This is the measurement of the process value that comes from the process:

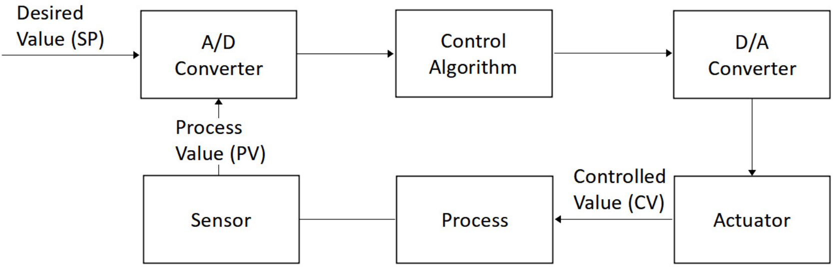

Desired
Value (SP)
A/D
Converter
Control
Algorithm
D/A
Converter
Process
Value (PV)
Sensor
Process
Controlled
Value (CV)
Actuator

Figure 3.2 – Measurement and actuator chain

Let's think about a very simple example. When you turn on your oven, the SP is the temperature that you want to set. The CV is the value of the electric current that feeds the heating elements of the oven, and the PV is the real temperature of the oven that is measured by the sensor.

Sensors

As we outlined in the previous chapters, the measuring device is actually made up of two main components:

- The sensor , which transforms the variable to be measured into the type of variable that is required for measurement
- The transducer , which accepts information in the form of a physical or chemical variable and converts it into a variable of a different nature (typically electric)

Very often, the sensor and transducer coincide in the same physical component. This is why the words sensor and transducer are often used interchangeably to indicate a device that measures a quantity and provides a signal, typically electrical, as an output.

A sensor is based on the identification of a physical law that binds two variables of different natures, such as temperature and resistance or speed and potential voltage. Therefore, there are many different types of sensors. You can distinguish between them according to the type of output supplied:

- Analog sensors : The output of analogical sensors are continuous values within a certain interval
- Digital sensors : The output of digital sensors can only be discrete values within a range
- Binary sensors : The output of binary sensors can only be one of two values: open or closed

A sensor has the following characteristics:

- Transduction : This describes the relationship between the input variable (the variable to be measured) and the output quantity. This relationship can be of direct proportionality, at least in the range specified for the input quantity. It is therefore either characterized by a proportionality constant, or it is non-linear.
- Bandwidth : This information is of fundamental importance in the

controller project, as it gives an indication of the maximum frequency at which the transducing characteristics of a linear sensor do not degrade. If this is much higher than the maximum frequency that is needed for a project, which is usually the frequency needed by the closed loop to be able to follow the dynamics of the process, the sensor is simply characterized by its own constant proportionality between the input and the output. Therefore, there is no need to consider a dynamic model.

- Dynamic features : These features are usually defined against the step response of the sensor. They include delay, rise time, time to half value, overshoot, and settling time.
- Input range : This refers to the range of variation of the input quantity for which the sensor guarantees certain properties, such as linearity, accuracy, or ace.
- Output range : This is the range of values that the output can be.
- Sensitivity : This is the ratio between the output and the input variation at a steady state.
- Resolution : This is the minimum variation of the input that produces a variation in the output and is therefore captured by the sensor.
- Linearity : This is usually defined as a range of input values for which the sensor has linear behavior with proportionality between the input and the output.
- Deterministic error (offset) : This is the systematic error that is produced at every measure. If it is known, this can usually be eliminated through sensor calibration.
- Probabilistic error : This is a random error, which may depend on

multiple factors.

- Accuracy : This term defines the maximum error that the sensor can make in a measurement operation. It is not to be confused with the resolution. Accuracy can be thought of as the sum of the deterministic and probabilistic errors.
- Precision : This is defined as the variance between several measures.

The following diagram highlights the difference between accuracy and precision:

    - Archer a has a high distance from the target (offset), but poor accuracy and precision.
    - Archer b has a high offset with low accuracy, but high precision.
    - Archer c has low offset and good accuracy, but low precision.
    - Archer d has a low offset and good accuracy. This means that they are accurate and precise.

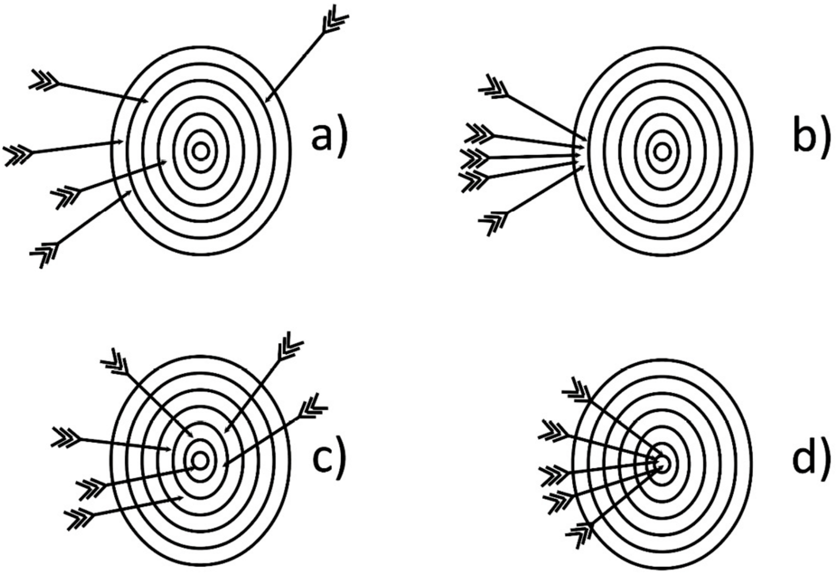


Figure 3.3 – Sensor offset: accuracy versus precision

- Hysteresis : This means that the output signal for a specific input can take several values, depending on whether the input variation is negative or positive.
- Drift : This can be defined as the variation of the output over long time periods when a constant signal is applied to the input.
- Electrical output impedance : This is useful to deal with problems related to interfacing. It's better to have a low impedance value since this helps couple the sensors to the device.
- Loading effect : This refers to the fact that the introduction of a sensor to a piece of equipment to measure a variable can alter the operation of the equipment itself.
- Noise : A sensor can generate noise in the output, which can corrupt

the detected information.

Converters

With the term conversion, we are referring to the transformation of some parameters of an electrical signal, while keeping the quantity of information from the signal the same. Converters are essential in control systems because they are present in any data acquisition and processing chain. Remember that an analogical signal can assume all of the values within a range, while a digital signal can assume only a finite number of values.

Digital to analogical

A Digital to Analogical Converter (DAC) acts as an interface between the numerical values that are generated by a calculation system in discrete time instants and the continuous time signals of the analogical world.

A DAC takes digital words as input and generates voltage or current values as output. It therefore carries out a transduction operation. If the converter memorizes the input word through an internal register, it must also directly carry out the so-called Zero-Order Hold (ZOH) operation. If it does not, the register must be preempted in the DAC when it interfaces with a calculator, even if it is not actually included.

The digital input word and the reference voltage control the output of a DAC according to the following relation:

$V_{out} = KV_{fs} (a_1 2^{-1} + a_2 2^{-2} ... + a_n 2^{-n}) + V_{offset}$

Here, Vout is the voltage in the output, K is the gain, Vfs is the full-scale value in the output, a_1 , a_2 , . . . a_n are the **n** bits of the binary world in the input, and Voffset is the offset voltage.

The output can be made available as a potential voltage or a current value, depending on the converter. The input word is generally handled as a binary fraction that has the decimal point on the left. a_1 is the Most Significant Bit (MSB) and a_n is the Least Significant Bit (LSB).

If K = 1 and $V_{offset} = 0$, the potential voltage in the output can also be represented as follows:

V o u t = N _ 2 n V f s

Here, N is the decimal value of the word in the input to the DAC. For example, the word *10001001* would be handled as the binary number *0.10001001* . Its decimal value, therefore, is as follows:

1 _ 2 + 1 _ 32 + 1 _ 256 = 0.53525625

Alternatively, you can consider the decimal representation of the input word. *137* is the decimal value that corresponds to the binary value *10001001* , while *256* is the decimal value that corresponds to the binary value *11111111* .

The reference voltage controls the full-scale value V_{fs} of the converter (or I_{fs} , if the output is in the current). Typical values of the potential voltage are 2.5, *5.0* , *10.00* , and *10.24* volts, which coincide with the required reference voltages. For outputs in the current, a typical value for the full scale is *20* mA.

The constant K represents the gain of the converter and is typically equal to 1. The offset voltage represents the output value when the digital input is zero and must be set to zero.

The resolution of a DAC is defined as the smallest change in output that is possible. It is determined by the number of bits of the input word and by the full-scale value of the voltage according to the following relation:

$$VoltResolution = Vfs\_2n$$

Here, n indicates the number of the bits of the converter. For example, a 12-bit converter with a full-scale voltage of five volts has a resolution of *1221* mV. This applies to mono-polar DACs, where the output is always positive. There are also bipolar

DACs, in which the input word is coded to represent negative values. In general, the relationship may not be linear because the input-output couples are not on a straight line, as shown in the following diagram:

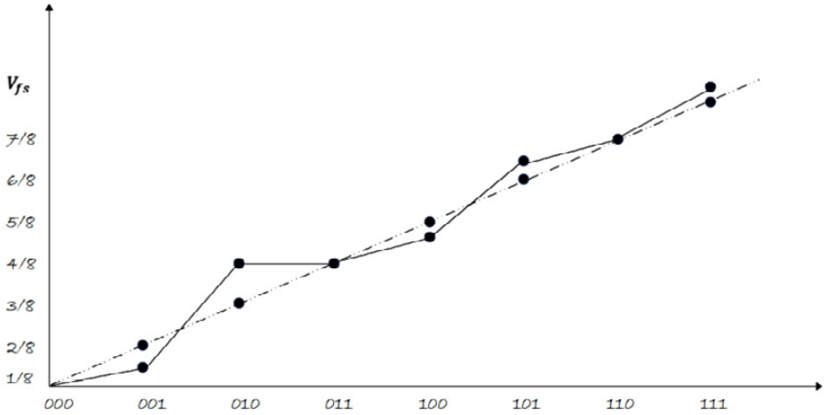


Figure 3.4 – Input/output in a real three-bit DAC

For this reason, some errors are defined against the ideal relationship that is represented by a straight line. These include the following:

- Error of linearity : This is defined as the maximum deviation of the output value from that of an ideal converter. It is usually expressed as a fraction of the LSB or as a percentage of the full scale.
- Error of differential linearity : When the input word changes by one bit, the output should vary by one LSB. If this does not occur, the differential linearity error is defined as the amplitude of the maximum

difference between the actual increase of the output when the input varies by one bit and the ideal increase.

- Error of integral linearity : For a given input word, the error of integral linearity is defined as the sum of differential linearity errors up to the given word.

DACs have a few important properties. These include the following:

- Monotonic : If the input word increases by one bit, the output must always increase, even if it does so by an incorrect value. This property is very important in a control system. If it is non-monotonic, this represents a variation of the desired phase, which can lead to unstable behavior.
- Settling time : This is the time needed for the output value to follow a change of the input word within a limited percentage error.
- Glitches : Glitches occur in the output due to non-simultaneous variations of the bits of the input word. The DACs on the market have specific circuits to minimize the occurrences of this phenomenon.

The performance of a DAC changes over time, with variations in the temperature and the supply voltage.

Analog to digital

An Analog to Digital Converter (ADC) acts as an interface between the continuous values of the analog world and the digital numerical values used in a calculation system. An ADC accepts an unknown analogical signal (typically, a voltage) and converts it into a digital word with `n` bits, representing the ratio between the input voltage and the full scale of the converter.

Generally, ADC converters include, or must be preceded by, a sampling and holding circuit to avoid changes in the input voltage during the conversion operation. The input-to-output relationship for an ideal three-bit monopolar converter is shown in the following diagram:

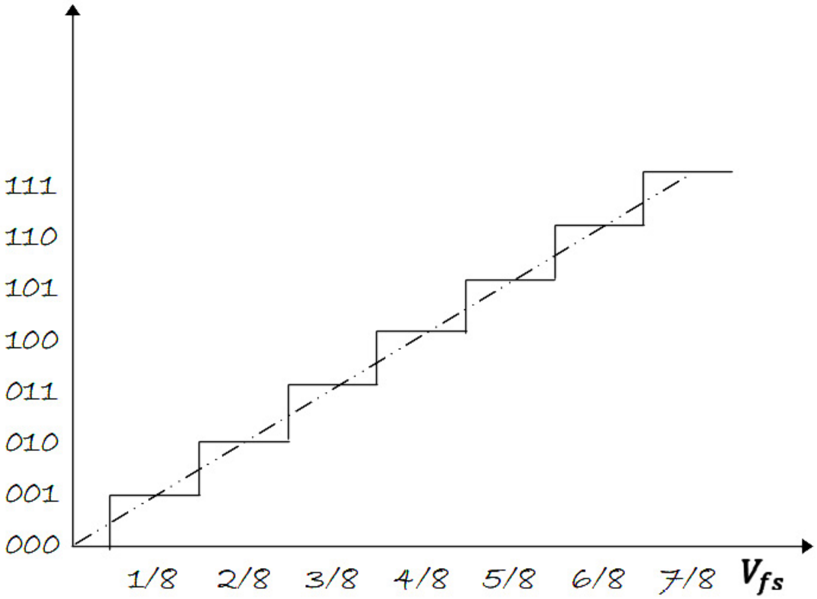


Figure 3.5 – Input/output in an ideal three-bit ADC

The output assumes that the encoded values are between *000* and *111* when the input varies between zero and Vfs. The amplitude of the code transition of the least significant bit corresponds to a variation of the input of V f s _ 2 n , which also indicates the converter resolution. Ideally, the code transitions should happen, for the three-bit converter example, at the values of V f s _ 16 , 3 V f s _ 16 , … .

The characteristics of an ADC can be expressed in the same way as the DAC. If an analogical voltage is applied to the input, the output of the ADC is an equivalent binary word: $V_{in} \approx V_{fs}(a_1 2^{-1} + a_2 2^{-2} + .. + a_n 2^{-n})$.

The approximation (≈) is necessary since the input voltage is reproduced from the binary word according to the converter's resolution. An alternative formula to calculate the binary word in output is given by the following equation:

$$N_{10} = int\left(\frac{V_{in}}{V_{fs}} 2^n\right)$$

Here, N_{10} is the decimal representation of the binary word.

The binary code must be steady for a change in the input voltage that is equal to 1 LSB. For an ideal converter, this means that when the input voltage increases, the output code should first underestimate the input voltage and then overestimate it. This error, called a quantization error , is represented in the following diagram:

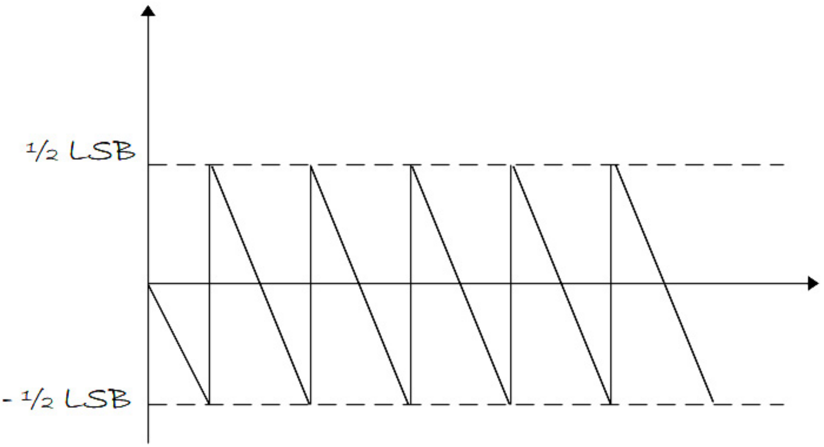


Figure 3.6 – Quantization error

The transitions of the encoded values in an ideal converter should be in a straight line. Due to quantization errors, this does not always happen, and we can define differential and integral linearity errors just like we did for the DAC. Each increment of the encoded values should correspond to a variation that's equivalent to the LSB of the voltage in input. The differential linearity error is the difference between the real increase of the input that causes an increase of one bit and the ideal increase corresponding to the LSB. The integral linearity error is the sum of the differential linearity errors.

For this type of converter, it is possible to define a condition of monotonicity, an offset error, and a gain error.

With regards to the representation of signals, we can have the following:

- Analogic signals : These can assume all the values within a range so that their values change continuously with time
- Quantized signals : These can assume a limited number of values according to the resolution
- Sampled signals : These are analogical signals that are evaluated in specific time instances and separated by the sampling time
- Digital signals : These are quantized and sampled signals that are coded as binary numbers

Actuators

In typical process control applications, the acquisition of the process variables to be controlled and the processing of the control laws is carried out through analogical or numerical controllers that work on a low power signal level. The variable that is processed by the controller and sent to the process to carry out the necessary corrective actions is handled in the same low-power representation. The process, on the other hand, may require high levels of power to be controlled. Examples of processes that require high levels of energy include thousands of cubic meters of a liquid or the forces of hundreds of thousands of Newton, such as in a metallurgical rolling process.

The function of the actuator is to convert low-energy control signals into actions of a power level that is suitable for the process to be controlled. This function can be considered as an amplification of the actuation signal generated by the controller. In many cases, the signal is also converted and then transduced into a completely different physical dimension.

A final control element carries out the necessary steps to convert the control signal generated by the controller into actions performed on the process. To change the flow rate of a fluid from 10 to 50 cubic meters per second, for example, a typical control signal of between 4 and 20 mA requires some intermediate operations. The necessary operations may depend on the process and the controller, but they can be summarized in the following diagram:

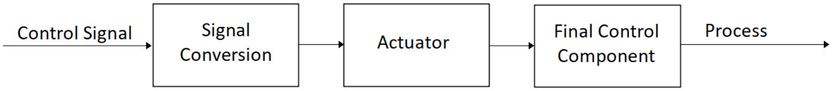


Figure 3.7 – Last control operation

The preceding diagram can be explained as follows:

- The first operation is signal conversion . This represents the modifications that must be implemented on the control signal so that it is compatible with the actuator. If, for example, the controller is numeric, and the final control element is a valve, the actuator may be a direct current motor. In this case, the signal conversion will be performed by a DAC and a power amplifier. The output of the conversion produces a transduced and amplified signal, which establishes the control variable of the process. This effect on the control variable is usually carried out by a physical device that is related to the process, such as a valve or a conveyor belt.
- The actuator can be defined as a device that converts the control signal into an action on the control component, not on the process.
- Finally, we arrive at the final control component . This device has a direct influence on the dynamic variables of the process and is an integral part of the process. If a flow rate must be controlled, for

example, the last control element might be a valve that is part of the fluid transport system. If a temperature must be controlled, the last control device will have a direct influence on the temperature.

Having established an understanding of the fundamental components, we will now investigate how controllers can control machines and components in an industrial scenario.

Controllers

We can define controllers as electronic devices that receive digital signals and process them according to a user-defined program for generating other digital signals in the output in a predefined time interval. In this section, we will explain the main types of controllers used in industrial plants: microcontrollers, PLCs, and DCSs.

Microcontrollers

A microcontroller is a programmable electronic device that is based on an integrated chip. It includes many of the components needed to make a control system. It can perform different functions autonomously, according to the program that's been implemented. From a computational point of view, a microcontroller is very similar to a microprocessor in that it has a Central Processing Unit (CPU) and executes control program instructions in data memory. It differs from a microprocessor, however, in its integrated functionalities. The main differences between microprocessors and microcontrollers are highlighted in the following table. The acronyms that are used are as follows:

- Electrically Erasable Programmable Read-Only Memory (EEPROM)
- Erasable Programmable Read-Only Memory (EPROM)
- Read-Only Memory (ROM)
- Random Access Memory (RAM)
- Arithmetic Logical Unit (ALU)

The differences between microcontrollers and microprocessors are as follows:

Microcontroller	Microprocessor
Microcontrollers run the program in a program is inside a special non-volatile memory area (ROM, EPROM, or FLASH).	In microprocessors, the program in an external memory area modified according to the CPU commands.
Microcontrollers have [illegible] memory for the RAM. EEPROM is also sometimes available for the data.	In microprocessors, [illegible] cache memory [illegible] the registers.
Microcontrollers decode the input and output ports, while logical levels are provided by external signals.	In microprocessors, the input and output ports are established by the device drivers.
In microcontrollers, A/D converters, comparators, and signal generators are integrated.	In microprocessors, functions are not integrated.
In microcontrollers, at least one timer is always present.	In microprocessors, the CPU clock is present.
Microcontrollers are Reduced Instruction Set Computing (**RISC**) devices.	In microprocessors, Complex Instruction Set Computing (**CISC**) is available.

Table 3.1 – Differences between microcontrollers and microprocessors

The microcontroller follows the Von Neumann architecture , in which the memory is unique to the system and the data. Compared to a control system architecture built with microprocessors and peripherals, an equivalent architecture based on microcontrollers reduces a lot of the complexity of the circuit diagrams since many of the services, such as memory and Input and Output (I/O), are included in the basic functionalities of the microcontroller. In a microcontroller, all of the main units communicate with each other through a BUS made up of 4, 8, 16, or 32 bits. This indirectly provides a measure of the capacity of the microcontroller to process this information. A microcontroller, like a microprocessor, recognizes only one type of language, called machine language . This has control instructions that are written using the hexadecimal system, which is not very human-readable. For this reason, a more intuitive language called assembler is used. The assembler language does not produce an executable file—just machine code.

Microcontrollers are used in systems such as the following:

- Alarm systems
- Automotive systems
- Electrical appliances (such as washing machines, ovens, and dishwashers)
- Distributor machines
- Control systems (such as temperature, pressure, and liquid level)

The main features of a microcontroller include the following:

- The management of a set of input and output lines
- Being able to process the data acquired in a binary format according to a program installed in non-volatile memory
- A timer, for the timing of the operations to be performed
- Programming in the assembler; the assembler differs from the chip vendor
- Programming in C and BASIC through specific libraries that are suitable for the microcontrollers

Depending on the type and the number of

integrated peripherals, the microcontrollers are divided into the following groups:

- Embedded microcontrollers
- Microcontrollers with external memory
- Digital Signal Processing (DSP)

Let's learn more about these in detail.

Embedded microcontrollers

Embedded microcontrollers have the largest integration capability. This is due to the following reasons:

- All of the components are internal to the controller
- They are used when high computing power is not required
- They are very cheap
- They are characterized by the following:
    - A medium- or low-power ALU
    - A small, built-in RAM, ROM, and EPROM-integrated I/O signals
    - Integrated timers

Microcontrollers with external memory

Microcontrollers with external memory are very similar to embedded controllers. They are also equipped with external memory so that they can manage greater complexity in design and maintenance operations, have a more complex and efficient ALU, and use the internal RAM, but also take advantage of external memory.

DSPs

DSPs are quite a recent development. Their power and computing capacity makes them very similar to microprocessors in the following ways:

- They are used to manage external and analog data.
- They are equipped with powerful ALUs and a set of ADCs.
- Generally speaking, they implement a small part of more complex systems. They act as interfaces for the output of these systems.
- They are used in digital controls.

PLCs

PLCs originated in 1969, when General Motors United provided the desired features for a new generation of controllers to be used in their production facilities (Walker M, Bissell C, Monk J. *The PLC: A Logical Development. Measurement and Control* . 2010). Currently, a PLC is based on a multiprocessor system, with network-integrated abilities that are able to perform very complex functions. It is based on almost the same technologies as a usual computer but adapted to the control of industrial processes. A PLC consists of several main components—a processor, I/O, a power supply, a remote I/O, a fieldbus, PID, a servo, and an encoder. It also contains other components that are available for specific uses as independent modules. The cabinet or rack contains and encloses all of the modules, ensuring the first deep integration from the pure mechanical world and the electronic world. There are some PLCs that are not modular but instead contain all of the fundamental components in a single device. The standardization of PLC programming languages from the IEC 61131-3 standard has greatly contributed to making PLCs more open and less tied to a specific vendor. The IEC 61131 standard provides the following five PLC programming languages:

- Sequential Functional Chart (SFC)
- Functional Block Diagram (FBD)
- Ladder diagram
- Instruction list
- Structured text

The first three are graphical programming languages, while the last two are textual languages. The most common PLC vendors in the market are Siemens AG, Rockwell Automation, Mitsubishi Electric Corporation, Schneider Electric, and Omron Corporation. In the following sections, we will look at some of the main modules of a PLC.

Processor module

The processor module is at the heart of the PLC system. It contains a board with one or more microprocessors that execute the operating system programs, the programs developed by the user, the memory where these programs are stored, and all of the components necessary for the operation. The processor module, as implemented by the operating system, provides a cycle that is composed of the following sequence of operations:

1. It updates the input memory area with the values coming from the physical inputs.
2. It executes the user program by operating on the values of the memory and always keeping the results in the memory.
3. It executes the system management programs, for example, for diagnostics.
4. It writes the physical outputs of the values that are stored in the reserved memory area.

The preceding sequence optimizes the communication of the input and output modules and guarantees that the stored values of the inputs remain unchanged during the execution of the programs. The reading of the inputs and the writing of the outputs is managed entirely by the operating system and not by the control program. The most common real-time operating systems are Microware OS-9, VxWorks, QNX, and Linux.

An important parameter of the processor module is the scan time . This is defined as the time interval that elapses between two consecutive activations of the same portion of the program application, thus comprising the time necessary to update the inputs and outputs. The scan time must not be confused with the response time of the system, which is defined as the maximum time interval between the detection of a certain event and the execution of the related programmed action. The response time also takes into account the delays introduced by the input and output modules.

I/O module

The I/O modules , both digital and analogic, are the modules through which the PLCs communicate with the physical process, detecting events and data from the sensors and ordering actions to the actuators. From an electrical point of view, they act as an interface between the voltage levels (TTL or CMOS) of the PLC circuits and the voltages (or currents) used for the transmission of the signals to the actuators. These signals range between a few tens and hundreds of volts. In most cases, this allows us to connect the PLCs directly to the various devices that are present in the field, minimizing the need for further signal conditioning. The analog I/O modules also carry out the digital-to-analogical or analogical-to-digital conversions that are required to directly interface the analog signals, which are continuous in time and value with the PLC. Being a calculator, the PLC works with strings of bits of a fixed length in discrete time intervals.

Remote I/O module

If the number of I/O signals is high and they are arranged over a large surface, it is convenient to carry out remote I/O. In this case, I/O cabinets are scattered in the industrial plant that is connected to the PLC through a remote I/O module that sends the status of the inputs and outputs of the cabinet, where the remote I/O is mounted through a serial line.

Network module

It is much more convenient to connect all of the sensors to the PLC through a common network or bus rather than connect each sensor through dedicated wiring. Industrial networks are specialized networks that ensure determinism in the message delivery process. The fieldbus modules manage the communication protocols for the various types of industrial networks, which include fieldbus, proprietary networks, and Ethernet.

Other modules

There also are additional modules for specific purposes. These include the following:

- Servo modules : They directly and independently perform the interlocking of one or more stepper motors, hydraulic motors, or direct current motors with incremental encoders.
- Encoder modules : They realize all of the functions that are necessary to use in one or more incremental or absolute encoders. These are essentially modules with high-speed counters.
- Proportional-Integral-Derivative (PID): The PLC provides them with just the required value or the SP for the process value. The PID autonomously calculates the value for which the controlled value fits the SP.
- Model Predictive Control (MPC): This is an advanced method of process control that is used to control the process while satisfying a set of constraints. MPC relies on dynamic models of the process, most often linear, empirical, or data-driven models obtained by system identification. See the following diagram:

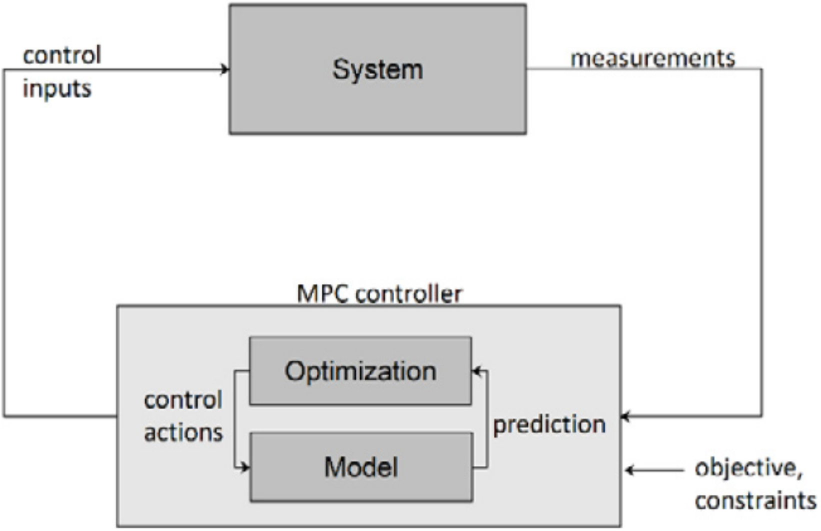

control inputs
System
measurements
MPC controller
Optimization
control actions
prediction
Model
objective, constraints

Figure 3.8 – MPC controller

Advanced Control

Advanced controllers or optimal controllers are a family of new controllers mainly based on innovative techniques such as machine learning and/or reinforcement learning algorithms. These controllers require high computational capabilities but can achieve better performance when the system cannot be modeled using linear models. A short list of advanced controllers is as follows:

- Data-driven surrogate model : Using machine learning or deep learning, we can build a surrogate model of the system based on data acquired during testing or operability; an optimizer leverages this model to take the next action

- Reinforcement Learning : An agent learns how to carry out an activity through repeated *trial-and-error* type interactions with the environment

The future of controllers

> *Machine learning-based controllers have demonstrated improved performance compared to MPC or PID. However, they require a simulation environment or a massive amount of data. Today, AI-based controllers are widely used in robot control, and they will be adopted in the industry more pervasively in the near future.*

DCS

In the field of industrial automation, DCS represents the most widely adopted solution for large continuous plants, including refineries, energy production plants, paper mills, glassworks, and chemical plants. The characteristics of the DCS include the possibility of distributing modules for data acquisition, processing, and control, and a communication network that is as efficient as possible between the various subsystems. Another important feature of DCS is its ability to change the network topology and add or remove modules on the fly with the system running. DCS can also perform functions that are normally implemented on PLC and SCADA in an integrated way.

The typical architecture of a DCS is made up of several levels. Starting from the bottom, we can find the interfaces to the field, consisting of the appropriate electronic acquisition boards (inputs) and commands (outputs). At this level, there are also communication interfaces for the most common fieldbuses, through which the information is exchanged with the transmitters and actuators that support the same type of protocol.

Through the I/O bus, the values of the inputs and outputs are conveyed to the controllers and processed according to the control strategies and the logic designed for the specific application. The software is developed through the implementation of function blocks, which therefore constitute the so-called system database. Each of these is labeled with a uniquely defined name, commonly known as a tag . It is therefore necessary to associate each database record with the relative tag and its fields to the different variables used in the function block, normally called items . A PID controller, for example, normally represents a database tag. Its items are the process variables, the SP, the control variable, the alarm thresholds, the proportional, integral, and derivative parameters, and so on.

Supervisors are above the controllers. Nowadays, they are usually implemented by Commercial Off-the-Shelf (COTS) operating systems. They act as an interface for the control room operator by accessing the tags and the database items, and consist of numerous screens through which a graphic representation of the system and the process is developed. This is one of the main differences between a DCS and a PLC-SCADA system. PLC and SCADA are two separate systems, each making use of their own variables and data structures, and exchanging data through a suitable communication driver, sometimes making use of a proprietary protocol. In the DCS, however, the database is unique, shared, and distributed between controllers and supervisors, each of which takes control of its processing tasks.

In general, a PLC controls one or more machining processes, while the DCS controls the entire plant. Typically, a DCS is used when the following conditions are met:

- The value of the product is very high. The production process is continuous.
- Failures in the system will result in a large money loss. The DCS normally has built-in redundancy, ensuring a higher level of system insurance.
- All upgrades are made online while the system runs continuously.

In any case, the differences between the PLC and

DCS have been reducing over the years. The most common DCS vendors on the market are ABB, Yokogawa, Honeywell, Emerson, and Siemens.

Now we know what controllers and operability are. In the next section, we will study how controllers communicate.

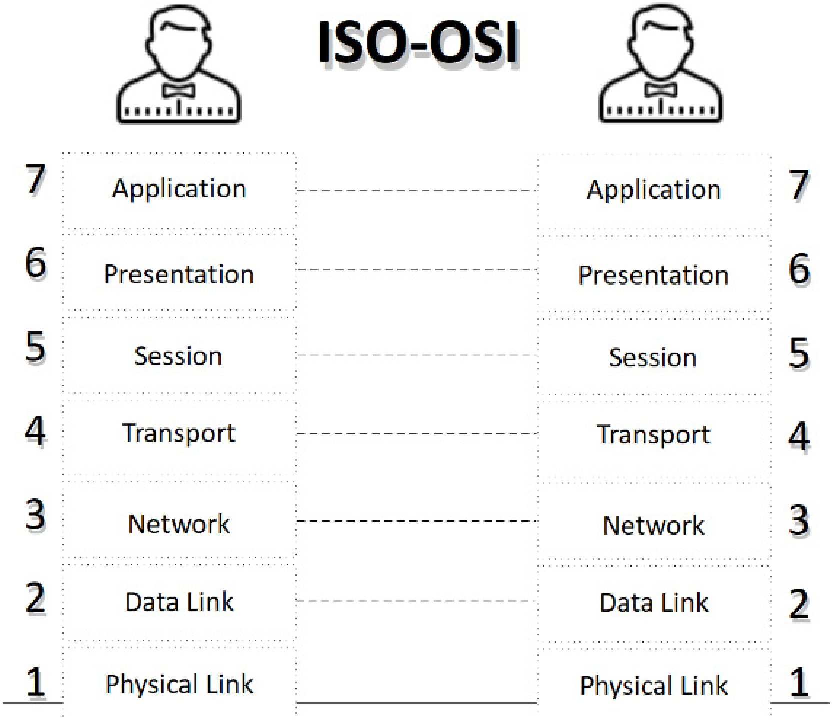
Industrial protocols

In an industrial scenario, we need to connect several components at different levels. Vendors have defined open and proprietary protocols to communicate efficiently. Despite the proliferation of industrial automation protocols, the Open Systems Interconnection (OSI)/ International Organization for Standardization (ISO) model provides a standard framework to interact with protocols.

The OSI/ISO model

The reference model for computer networks is the

OSI model, which was developed by the ISO. It is a conceptual model to which the computer networks' makers can refer rather than a precise description of a real network. In the ISO/OSI model, every node of the network can be sketched as a hierarchical structure of seven levels, as shown in the following diagram:



ISO-OSI
7
Application
Application
7
6
Presentation
Presentation
6
5
Session
Session
5
4
Transport
Transport
4
3
Network
Network
3
2
Data Link
Data Link
2
1
Physical Link
Physical Link
1

Figure 3.9 – Functional layers in the ISO/OSI model

The functional layers in the OSI model are explained as follows:

- The first level, starting from the bottom, is level 1, or the physical level. This includes the mechanical and electrical connections between the nodes, together with the software drivers for the communication ports. The features of this level establish some parameters of the network, such as the speed and the transmission mechanisms.
- The second level is the data link. Like all of the following levels, it is implemented in the network software of each node. This level is responsible for the exactness of the flow of the structured sequences of bits that are exchanged between the nodes. These are also called frames . These frames are composed of the information to be transmitted, plus a control code that is added at the beginning and checked upon arrival. If an error occurs, an attempt to restore the right sequence is carried out using the control code that was added. If the attempt fails, the re-transmission of the corrupted sequence of bits is requested. The primary responsibility of this level is to ensure an error-free connection between the nodes.
- The third level, the network level, guarantees a logical path between two nodes that are not directly connected. This level defines the characteristics of the network from a management perspective, such as the node addresses, the regulation of the accesses, and the treatment of the collisions. In summary, this level implements the routing and the interconnections between the nodes.
- Level 4, the transport level, independently implements the transport functions of the network structures of the underlying lower levels, ensuring the transfer and the integrity of the messages and their re-

transmission, if necessary. It acts as an interface between the network and the application software of the next three levels.

- Starting at level 5, the session level, we enter the user area. This level manages the ordinary data exchange and the synchronism between the nodes, making the remote connection possible. This level therefore includes the functionalities necessary to support the exchange of information between different machines.
- Level 6, the presentation level, is the level where the information is encoded and vice versa. The sequences of the binary data are related to their meaning and represented in the form of texts, figures, and so on.
- Level 7, the application level, provides interfaces and services to application programs, such as file transfers between nodes, distributed databases, web applications, and so on.

The following are some noteworthy features of the ISO/OSI model:

- The real physical connection between the nodes only exists at the first level. For all other levels, there is a virtual connection.
- Each level is responsible for a defined set of functions such as coding, fragmentation, or routing. These are implemented by specific entities or functional blocks.
- Each level can interact with the level below and above on the same node. Each level can also interact with the same level on other nodes.
- In passing the sequences of bits (data packets) from one level to a lower one, a piece of information is added to mark and identify it. This is known as a Protocol Data Unit (PDU).

- Each level adds its own PDU to the information received from an above level and removes it if the information comes from a level below. This is known as an encapsulation technique.

The main features that differentiate these computer networks are their topology, the transmission media, and the mechanism or protocol for accessing the network. The topology defines the physical schema of the network, or how the different nodes are connected to each other. The following are some examples of topologies:

- An example of a topology is the bus or open loop . This topology does not have any problem adding or deleting nodes, since the information moves independently, but there is only one possible route between the nodes.
- Another example is the ring or closed loop , in which the messages cross the nodes. In this case, the latter must be able to identify which node the receiver of the message is and eventually send it back.
- A further example is the star topology , whereby a primary node is connected directly to all other nodes. All messages have to go through the primary node.

> ***Note***

> *Hybrid topologies are also possible. For example, it's possible to implement a star topology wherein one component is connected to a ring topology.*

Transmission media defines the physical support through which the information flows. Its characteristics obviously have a direct influence on network performance. The simplest transmission media is the twisted pair, which can either be shielded or not. Other transmission media include the following:

- Coaxial cable : This is more difficult to install, but it does support multiple modulated transmission channels at different frequencies due to its bandwidth.
- Optical fiber : This is the best transmission medium in terms of its transmission rate, immunity to noise, and the possibility of having multiple channels inside the same physical media. However, it is more expensive and its installation is more hardworking.
- Powerline : These use low powerlines from between 230 and 400 V to transmit a frequency-modulated signal. They can reach speeds of a few hundred KBps, but there is often noise on the network.
- Wireless networks : These use infrared frequencies and industrial field radio frequencies. They do not require wiring, but they suffer from interference problems and noise.

The access method defines how the nodes can access the network and avoid transmission conflicts. Some access methods are explained as follows:

- Centralized access or the polling method : In this method, a master node systematically interrogates other nodes. It is a deterministic, simple, and reliable method that is easy to manage and implement, but it neither allows high transmission speeds nor the prioritizing of the accesses.
- Token access method : In this method, a specific sequence of bits called a token is sequentially transferred between all of the nodes that are connected to a network with a ring topology, or a token ring. When a station receives the token, it re-transmits it and adds the message within a predetermined time interval. This is a deterministic access method. The ring can be also logical rather than physical and therefore reconfigurable. In this case, we refer to it as a token bus.
- Carrier Sense Multiple Access/Collision Detection (CSMA/CD): According to this method, each node listens to the current carrier on the transmission media to ensure that the channel is free before transmission. This does not guarantee that there are no collisions. During the transmission, the station keeps listening to detect any collisions; if a collision occurs, the message is re-transmitted after a randomly determined time interval. The main problem with this type of access method is the lack of determinism. Its efficiency also depends on the number of connected nodes.
- Time division method : In this method, a time range is cyclically assigned to each node to transmit its information. This method ensures full determinism, but it is difficult to implement because you

need to have time synchronization between all of the nodes. In addition, sometimes it is assigned to a node to transmit, even if it does not have any communication targets.

- Producer/consumer access method : In this method, one of the nodes is the broker of the network. The nodes are identified either as producers of a specific set of information or consumers of that information. The broker of the network, rather than enabling a node to transmit or preventing it from doing so, allows the communication of a specific set of information implicitly by allowing the node producer to transmit it and the consumers to receive it. The sequence in which the information is enabled to be transferred determines the time needed to update the information, therefore ensuring the determinism of the network.

Automation networks

Looking at the CIM model, we can understand how important information being exchanged between devices on the same or different levels is. In the lower levels, we have a large amount of simple information that is exchanged within small and determined time intervals. At higher levels, we need to transfer less information that is more complex and within longer time intervals that are not necessarily determined. In an integrated production system, different types of IT networks are needed, each tailored to the task to which it is dedicated.

We can identify three categories of networks—information network, control network, and field network, as follows:

- Information network : This ensures the communication between the control devices and the applications and devices that manage the plant and the company. It is a network that must be able to process complex information that is composed of many bytes. This information is exchanged with low frequencies and there is no need to ensure that the message is delivered within determined time intervals. It is common to use Ethernet for the first two levels of the OSI model and TCP/IP protocols for the third and fourth.
- Control network : This must ensure the communication between the devices dedicated to the control and supervision of the plants. The information that's sent is not very complex, but the transmission has to be within determined time intervals and at higher frequencies. The control networks are usually developed by the PLC makers, and their implementation is proprietary. This means that the other devices in these networks can talk with the PLCs through specific network cards and programs for interfacing. For these networks, the first two levels of the OSI model are typically based on a token to ensure determinism in the transmission. The software for network management is usually integrated into the system provided by the PLC maker, which allows the user to configure the network by configuring the physical

connections between the nodes by assigning a logical address to each of them. Over the last few years, the market has been driving a standardization process for the control networks through formal or de facto standards and also using Ethernet for the first two levels of the OSI model.

- Field network : The networks dedicated to the field (or fieldbuses, as they are commonly referred to) have been introduced to act as interfaces for the smart sensors and actuators to the control devices, as they have basic computation capabilities. We will discuss fieldbuses further in the next section.

The fieldbus

Field or fieldbus networks are computer network implementations for connecting control devices, such as PLCs, and devices on the workshop or field, such as sensors or actuators. In a typical scenario, sensors and actuators are connected to the controller either directly, through the input or output modules of its cabinet, or through a serial line from the remote input and output cabinets. Using a field network, however, such devices become the nodes of a computer network and so they must have some processing capacity to be able to communicate through the network.

These networks differ from the preceding networks because of the following points:

- They have very short messages that are exchanged frequently.
- They deliver the messages within a short and deterministic time interval.
- They address the messages to many nodes simultaneously. An example of a message of this kind might be a closing order set to all of the valves of a system.

These characteristics lead to the creation of lean networks, where just levels 1, 2, and 7 of the OSI model are defined, leaving the intermediate levels empty. This reduced model of the OSI model is called an Enhanced Performance Architecture (EPA) model, and is shown in the following diagram:

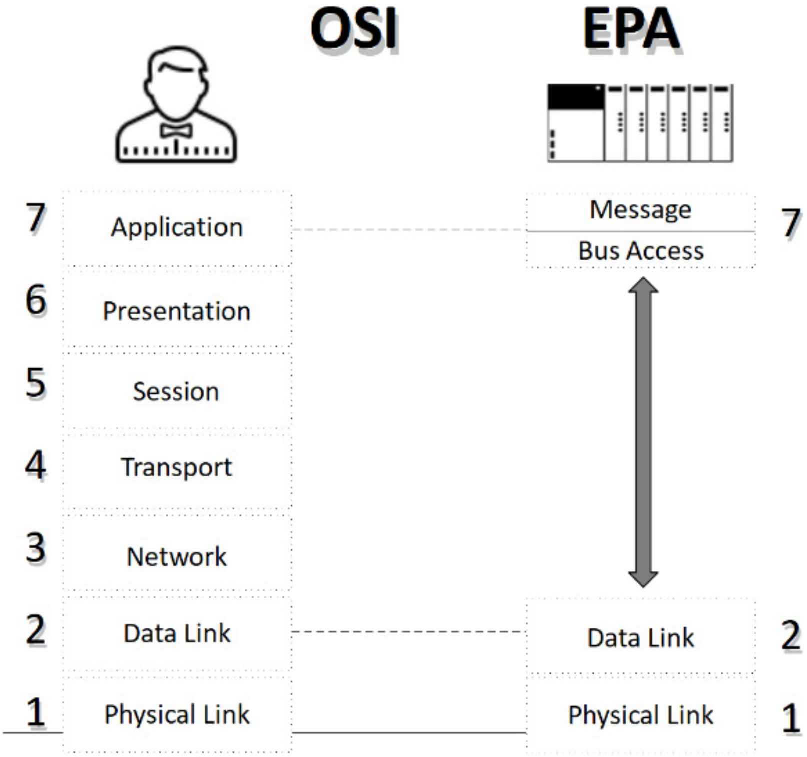

OSI
EPA
7
Application
6
Presentation
5
Session
4
Transport
3
Network
2
Data Link
1
Physical Link
Message
Bus Access
7
Data Link
2
Physical Link
1

Figure 3.10 – ISO/OSI and EPA model

Fieldbus networks provide the following advantages:

- They have a simplified architecture since they are scalable and flexible.
- They have less wiring, with a reduction in installation and maintenance costs.
- They can handle more complex and bidirectional information since the fieldbus devices can operate signal processing locally. This allows the fieldbus devices to do the following:
    - Linearize or convert the signals into engineering units

    - Send information about their state to the controller
    - Carry out close-loop regulations

- They have a local processing capability. This makes the response time less critical since many of the necessary operations, including the closed-loop regulations, are carried out by the device itself autonomously.
- They can carry out calibrations of sensors and actuators through software from a terminal connected to the network.
- They can provide robustness in their transmissions since digital transmissions are intrinsically less sensitive to noise than analogical transmissions. They can also implement techniques for the recognition and correction of transmission errors.

Even if all of the devices that require networks such as these are located at the lower level of the CIM hierarchy (on the field level), they need different functionalities. For this reason, there are three different classes of field networks—the sensor bus operating at the bit level, the device bus operating at the byte level, and the control bus operating at the level of blocks of bytes. This is depicted in the following diagram:

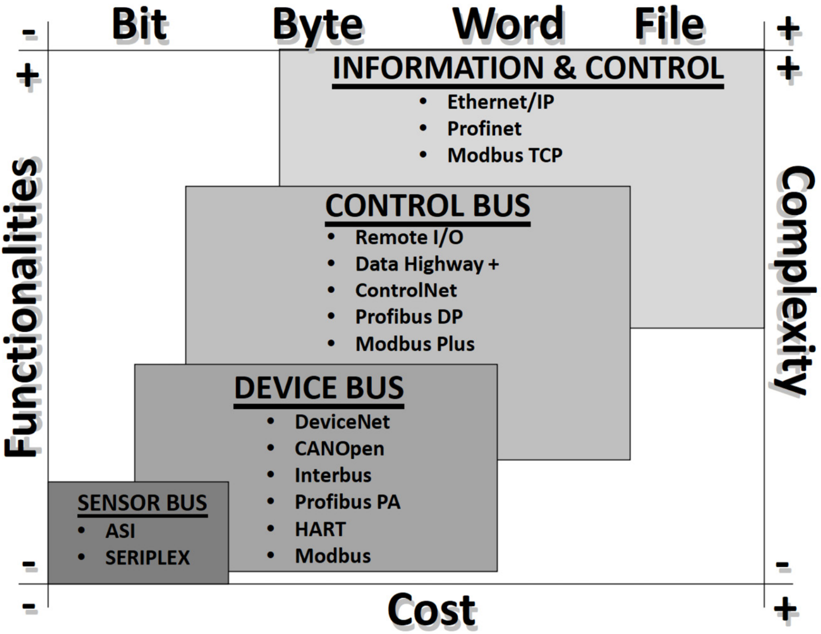

Bit
Byte
Word
File
-
+
+
+
Functionalities
Complexity
INFORMATION & CONTROL
• Ethernet/IP
• Profinet
• Modbus TCP
CONTROL BUS
• Remote I/O
• Data Highway +
• ControlNet
• Profibus DP
• Modbus Plus
DEVICE BUS
• DeviceNet
• CANOpen
• Interbus
• Profibus PA
• HART
• Modbus
SENSOR BUS
• ASI
• SERIPLEX
-
-
-
Cost
+

Figure 3.11 – Fieldbus layers and protocols

Therefore, let's explore the details.

- Sensor bus : This sensor bus usually implements just the first two levels of the OSI model, that is, the physical and data links. Its primary purpose is to reduce the wiring; the typical length of a message is one byte. Usually, the devices that are already available are connected to the network through a multiplexer. The most common sensor buses are CAN, Seriplex, ASI, and LonWorks. An example of a connected device could be a proximity sensor without internal diagnostics.
- Device bus : This allows the transfer of messages that are up to 16-32 bytes in size. In addition to the first two OSI levels, it implements some features belonging to level 7, the application level. Device buses also allow us to send a simple diagnostic. The most common device buses are CAN, DeviceNet, Profibus DP, LonWorks, SDS, and

Interbus-S. An example of a connected device could be a photocell or a temperature sensor with internal diagnostic functions.

- Control bus : This allows for the communication of blocks of bytes, which can be up to several thousand bytes. It implements levels 1, 2, and 7 of the OSI model, as well as an additional level called the user level. The devices developed for connecting to a control bus include predefined algorithms that are configured through the network so that they can adapt them to a specific application. They also include a real-time internal cache that checks and updates all data continuously, making it available to all other devices on the network. An example of a connected device could be a valve with an embedded flow regulator, as well as self-diagnosis and configuration capabilities. The most popular control buses are WorldFIP, Profibus, Modbus, and DeviceNet.

In this section, we have explored the basic concepts of the industrial protocols; in the next section, we will discover how to communicate from a central system.

Supervisory control and data acquisition (SCADA)

SCADA is a software application for the supervision and control of automated processes from a centralized location, both locally and remotely. The heart of a SCADA system is its modular and distributed software architecture. Control systems, which are typically (but not always) PLCs, have the responsibility of feeding the SCADA system with the values of the process variables to monitor the machine and the process behavior. The SCADA server nodes have the necessary drivers to communicate with the field devices either through their legacy communication protocols or through OPC or a fieldbus protocol. At the server level, the data is gathered on a regular basis or, when changes occur, processed to filter noise, and then stored in a real-time database. There are two different types of data collection:

- Polling : The signal is sampled on a regular basis. The polling time is the time interval between the sampled values of the same signal. Even so, the sample of an analogical signal is only gathered by SCADA if its value changes by more than a predefined dead-band compared to the value of the previous sample collected.
- Unsolicited : The data is gathered when its value changes. The data collection interval is the shortest time interval in which a variation of

the value of the signal is caught by the data acquisition system.

The data acquisition interval depends on the following factors:

- The time taken for the communication channel and the related protocol to talk to the device from which the data is being gathered affects the data acquisition interval. For instance, if the message brings a lot of information, the buffer can be overloaded, and the sampling time can be delayed.
- The performance of the driver in terms of how long it takes to refresh the data of the device from which the data is gathered also has an impact.
- The amount of data to be collected from the same driver and data source also impacts this.

Typical data acquisition intervals for SCADA applications are in a range of between half a second and one minute. It is not very common for SCADA to carry out data acquisition sampling in under half a second unless you have dedicated hardware specifically for this purpose.

Client nodes are mainly dedicated to data visualization. The communication between the server and the client nodes is typically based on the

Ethernet protocol. The presence of multiple server nodes makes it possible to process data in a distributed way.

The core of a SCADA system is the process database to which all other modules that make up the system refer. A small or medium-sized control system typically handles thousands of process signals. It adopts a common approach to processing information based on a set of parameters that are necessary for the correct management of the information.

These include the tag name, description, sampling time, minimum or maximum value, engineering units, and so on. This piece of information is based on the asset registry of the device to which the signals belong. The asset registry and the related asset model are very important for industrial companies because they are used for measuring performance, planning maintenance, and designing, setting, and running analytics. We will explain these in detail later. The user interface of a SCADA system allows the operator to access the tag name database and send commands to the system through graphic pages and synoptic panels of either the whole system or parts of it. This is also called the HMI. Given the importance of the graphic representations that are used in the operator user interface, SCADA systems provide a library of graphic objects that allow you to develop complex synoptics easily.

SCADA also allows you to set thresholds and configure events and conditions for detecting anomalies and generating alarms. Alarms are classified by type and priority.

Another important task that is carried out by the SCADA system is tracking production sequences and volumes. It does this both for past data, by using the stored time series of the values of the process variables, and for the current moment, through the real-time graphical representation of the evolution of these values. The storage of this information takes place either through specialized databases for managing time series (such as Historian) or through relational databases that are managed directly by the SCADA application.

SCADA applications provide both dedicated programming languages and common scripting languages such as VBScript or JavaScript to program and customize tasks according to the requirements of the process. They also often provide integration capabilities with the most common programming frameworks such as .NET or Java. Some of the most common SCADA applications on the market are iFiX, InTouch, WinCC, FactoryTalk, and Iconics.

SCADA allows us to efficiently connect the devices and machines in a central system; in the next section, we will see how to store this information.

Historian

In the previous sections, we saw how data acquisition in SCADA systems starts from the controllers (the PLCs or the DCSs) or RTUs gathering measurements from sensors and equipment through industrial protocols. The digital representations of these measures are usually called tags or datapoints. Each of these represents a single input or output signal that is monitored or controlled by the system. They usually appear as value-timestamp pairs. After generation, the data can also be sent to other monitoring servers for analysis by humans or to the MES system for planning and maintenance. At the same time, data often feeds a specialized database for storing and managing times series. This specialized database is called Historian, or data Historian . Historians are not relational or NoSQL databases; they have fewer capabilities and features and a much simpler structure.

However, they are designed and optimized for managing situations related to time series. They can acquire data from the controllers, ingest a huge amount of data, optimize storage, and retrieve data quickly.

Historians developed and evolved following the growth of automation in industrial processes. Since production processes have become more automated, controllers have become more powerful, industrial protocols have become more standardized, and data and information have become more available, there is a greater need to collect, store, manage, process, and retrieve industrial signals in industrial plants. In a typical small company, there may be one Historian, while larger companies are likely to have a Historian in each plant of the company and another one at the company headquarters to collect the data that comes from all of the plants.

Up until a few years ago, the cloud was not a real option for databases such as this. As Historians acquire data from controllers and require a high throughput in both input and output, they need to be close to their data sources, and so they generally lived on premises. Recently, however, a generation of cloud-native Time Series Databases (TSDBs) have been developed to store time series in the cloud (see *Chapter 7*). Over the next few years, they may well replace on-premise Historians, or, more likely, be strictly integrated with them. We may well have a scenario in which the Historian on premises acts as a data collector and temporary storage, and pushes all data to a cloud-native TSDB into the cloud. In the following few paragraphs, we will just refer to on-premises Historians.

Historians need to gather data from controllers. They implement software interfaces for the most common industrial protocols and fieldbuses, including Profibus, Modbus, DeviceNet, and OPC, and sometimes also legacy protocols for old plants or DCS. Connectivity to industrial devices is one of their most important features. Some Historians implement just a tenth of the industrial network protocols, while others, such as the OSI-PI, implement much more. These interfaces implement the most efficient way to gather the data according to the capabilities provided by the specific protocol. Data acquisition by polling is always supported, as is the dead band mechanism. Unsolicited data collection depends on the capabilities provided by the underlying protocol. In addition, the insertion of values that don't come directly from the controllers but are instead calculated or derived by them can be carried out through specific interfaces or APIs.

Historians provide fast insertion rates: up to tens of thousands of tags are processed per second. These performances are made possible by a specific buffer area that keeps recent values in memory, sorted by increasing timestamps, to later write the data on the disk.

Historians also provide a functionality called store and forward , which prevents data loss during planned or unplanned network outages between the data sources and the Historian server. To store large amounts of data with minimum disk usage and acceptable approximation errors, Historians often rely on efficient data filtering or lossless compression engines. The data filtering mechanism ignores the incremental changes in values that fall within a deadband that is centered around the last reported value. It only takes into account new values that fall outside the deadband and then centers the deadband around the new value. Basically, it identifies and discharges repeating elements.

The data compression mechanism is applied to all values that pass through the data filtering mechanism. Its implementation depends on the specific Historian product, but the basic idea behind it is to identify and discharge the redundant data to avoid losing content from an information perspective. The data compression algorithms hold the last one or two values in memory, building up a sort of swinging door and storing or discarding the values based on the next value acquired. If the swinging that's achieved by replacing the hold value with the last acquired value allows us to avoid losing pieces of information, the held values are discarded. The loss of information related to the discharge of the hold values depends on the specific Historian product, and it is a trade-off between the storage allocation and the amount of information coming from the discharge. It can be tuned by the user through specific settings. The values that are physically stored at the end are typically called raw data .

Historians are key components in information systems, providing data for plant operability and business intelligence, through time series-specific features such as interpolation, re-sampling, or retrieving pre-computed values from raw data.

Interpolation is a very important feature that differentiates Historians from other types of databases. The raw data is stored in the Historians according to the data collection interval and depending on their variations. Tags that have the same data collection interval have values that are stored in different timestamps. The same happens for tags that have their data collection configured as unsolicited. This means that the signals that are related to the same physical asset (for instance, a tank) have their raw data stored with different timestamps, meaning that if we need to get a snapshot of the status of that asset at a specific time, most of its signals do not have raw data for that specific time. The value is provided by Historian through interpolation. This involves different algorithms rebuilding the missing data using the raw data and also providing an indication of how accurate the interpolation is. Values that are not measured directly, including key performance indicators or diagnostics, may be computed automatically according to the input flow and then stored in the Historian archives.

Some Historians store time series; time series can be organized in a hierarchical data model, which normally reflects the operating plant. This data model refers to the asset model and is consistent throughout the plant's organization to ease browsing and so that you can group similar time series by subsystem. Sometimes, the asset model is a separate module within the Historians application. Other times, it is an external application or module that is part of the MES system to which the tags are linked. In either case, the asset model is a fundamental piece of the industrial data architecture, as we will see in the section on MES later.

Visualization features are provided by standalone services or web applications that are supplied with Historians. They facilitate a sophisticated exploration of the archived data by displaying plots, tables, and time series trends that retrieve only the representative inflection points for a considered time range, statistics, or other synoptics. Historians also provide a proprietary SQL interface, a sort of SQL dialect that is used with legacy extensions to support their specific features.

Roughly speaking, Historians are characterized as follows:

- A simple schema structure, based on tags with no support for the transactions
- A SQL interface to query the archived data, offering ad hoc features for time series
- Connectivity to the industrial protocols to gather data from the controllers
- Data filtering and compression mechanisms to optimize bandwidth and storage consumption
- A store and forward mechanism, which makes them resilient against the instability of network connectivity
- The ability to manage a high volume of append-only data
- A replica mechanism, often called Historian-to-Historian , for pushing streaming data between two instances of the same Historian product

- Data row interpolation and built-in capabilities for managing industrial data Visualization capabilities and asset model support

From an architectural perspective, Historians can be split into two main categories:

- Historians that use a legacy architecture for storing the data make up the first category. In this category, we have OSI-PI (OSIsoft), Proficy Historian (GE), and InfoPlus21 (AspenTech).
- Historians that use a third party such as Microsoft SQL Server for storing the data make up the second category. In this category, we have Wonderware Historian (Schneider), FactoryTalk Historian (Rockwell Automation), PHD (Honeywell), and SIMATIC IT Historian (Siemens).

Very often, the companies that develop controllers offer a product suite where each product covers a specific area of the plant's automation system, while at the same time providing an integrated and unified interface to the users. This means that, in the same suite, there are tools to develop, test, and download the control program, a SCADA system, a Historian application, a tool for designing and maintaining the asset model, and other tools and applications that make up the MES system. Some of these include the following:

- FactoryTalk (Rockwell Automation)
- SIMATIC IT (Siemens)
- Proficy (GE)
- aspenONE (Aspentech)
- Wonderware (Schneider)
- PI System (OSIsoft)

Historians are the *most common database* of an industrial plant. In the next section, we will go through ERP and MES to understand how to integrate all the previous information in a single industrial system.

ERP and MES

ERP is an information system that integrates all of the relevant business processes of a company, including sales, sourcing, finance, warehouse management, logistics, transportation, accounting, and so on. This allows the business to manage and keep the whole company under control. If there is an order request, for example, the sales manager can check in real time whether a given product is in stock, confirm its availability, and estimate the delivery time. If the order is confirmed, this can be automatically passed to the logistics department, thereby avoiding wasting time and paperwork and bypassing other useless intermediate steps. Nowadays, modern ERP systems cover all areas that can be automated and/or monitored within a company, thus allowing users to operate in a uniform and integrated context, regardless of the application area.

The term MES refers to an information system that has the responsibility of managing and controlling the production execution functions of a company. The main features of an MES system are as follows:

- Standard interfaces to keep the ERP, warehouse, and field in sync according to the ISA-95 and ISA-88 standards are included. This includes ERP orders, material, product definitions, planned production schedules, production performances, and material consumption.

- Another feature includes traceability, backward traceability, and full identification of all of the materials in the plant using radiofrequency identification and bar code scanners.
- The management and execution of orders that are created manually within the MES solution or that come from the ERP level is a further feature. These include production orders, maintenance orders, transport orders, and so on.
- Another feature is material data management and product definitions that are created manually within the MES solution.
- The integration of data that comes from multiple systems into a database optimized for Online Analytical Processing (OLAP) is another MES system feature. This data might come from Laboratory Information Management Systems (LIMSs), statistical processes, SPC, ERP, or SCADA.
- Another feature is the monitoring of the production lines and equipment with the option to associate the reasons for the interruptions of the production activities.
- Compliance with standards and regulations and product quality is a further feature.
- Stock accounting, material inventory and consumption, and work progress in terms of quantity and time are also included.

MES provides information that helps production managers understand which actions to put in place to optimize and improve production. MES works in real time (from the user's perspective) to keep the production process under control. This includes materials, personnel, machines, and support services. Moreover, because you are able to capture data from the control system, you can use MES to create an in-built record of your manufacturing processes. This is particularly important in regulated sectors such as food, beverages, or pharmaceuticals. MES is an intermediate system between ERP and SCADA, also known as a process control system. The implementation of an MES system provides the following advantages:

- Reduction of waste and rework
- Faster setup times
- More accuracy in calculating the costs related to production
- More effective production flow
- Inventory reduction by eliminating just-in-case stocks
- Implementation of a paperless activity workflow

The asset model

A key element of any automated production system is the asset model. This is the conceptual model of the physical or logical assets that a company uses to perform its business. It typically lives in the MES system and is organized hierarchically according to the ISA-95/ISA-88 standard. The ISA-95/ISA-88 standard provides a common model for production processes and resources, as well as a standard interface to expose the model to higher-level ERP systems and lower-level control systems. The ISA-95 standard was originally developed to bridge the gap between low-level machine control and high-level ERP systems that are responsible for planning and scheduling. Currently, ISA-95 is the international standard for the integration of enterprise and control systems. It consists of both models and terminology. These can be used to determine what information has to be exchanged between systems for sales, finance, and logistics and systems for production, maintenance, and quality. This information is structured in UML models, which are the basis for the development of standard interfaces between ERP and MES systems. ISA-95 is based on the hierarchical structuring of ISA-88 for the physical assets of the industrial company. While ISA-88 focuses on the process cell, ISA-95 focuses on a higher level. For ISA-95, the levels of the process cell and lower are only of interest if it is necessary to exchange information between the enterprise and control systems regarding these levels.

The ISA-95 equipment model is a standard that describes the logical entities; ISA-95 is represented in a hierarchical way to model the aspects of production. The general structure of the ISA-95 equipment model is represented in the following diagram:

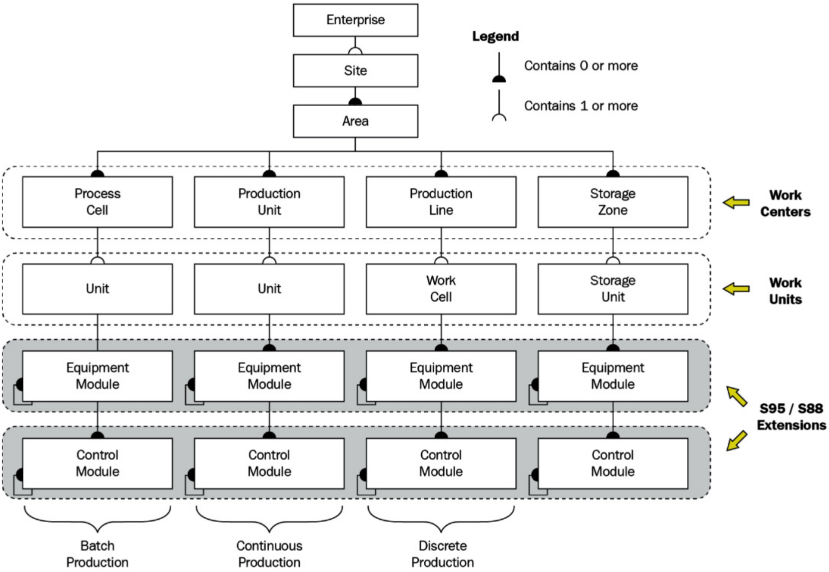

Enterprise
Site
Area
Legend
Contains 0 or more
Contains 1 or more
Process Cell
Production Unit
Production Line
Storage Zone
Work Centers
Unit
Unit
Work Cell
Storage Unit
Work Units
Equipment Module
Equipment Module
Equipment Module
Equipment Module
S95 / S88 Extensions
Control Module
Control Module
Control Module
Control Module
Batch Production
Continuous Production
Discrete Production

Figure 3.12 – S95/S88 model – ANSI/ISA-95.00.01-2000: Enterprise-Control System Integration. Part 1: Models and terminology, 2000.

This model has a number of important characteristics. These include the following:

- There are similarities between the process cell, the production line, and the production unit. These are essentially the same levels of entity that are applied to batch, continuous, or discrete production.
- Storage zones and units do not depend on the type of production.
- Equipment modules can contain other equipment modules, and control modules can contain other control modules. This allows you to build a complex hierarchical model of a piece of equipment.
- Equipment modules and control modules are not part of the basic

ISA-95 standard. Indeed, they have been added as extensions of the original ISA-95 model to recombine the model with the ISA-88 standard, which supports these components.

ISA-95 equipment entities

Figure 3 .12 describes the various equipment entities and their relationships in the ISA-95 model. These entities are described in detail as follows:

- Enterprise : An enterprise is an organization that has a definite mission, goals, and objectives. It could represent an entire company, a division within a company, or a functional component of a company dedicated to a product or service. Corporate, divisional, and business unit activities are decided at the enterprise level.
- Site : A site is generally a physical or geographic grouping of a portion of an enterprise. A factory could be a site, an entire complex containing multiple factories, or a single building. The exact determination of a site depends on the size of the enterprise and the organization of the various activities within it.
- Area : An area is a physical, geographical, or logical grouping that's determined by the site. It can contain process cells, production units, production lines, and storage zones.
- Work center : A work center is a facility that consists of resources in a role-based hierarchy. It performs production, storage, material movement, and any other self-contained activity.

- Production line : A production line is a work line that includes equipment required for the manufacturing of a discrete product or a family of products. A production line must include all of the equipment required to build a discrete product or a component of a product.
- Production unit : A production unit is a work center that contains production equipment that converts, separates, or reacts with one or more feedstocks to produce intermediate or final products.
- Process cell : A process cell is a work center that contains production equipment to carry out a processing step in batch operations.
- Storage zone : A storage zone is a work center that serves as a comprehensive inventory subsystem, equipped to receive, store, retrieve, transfer, and ship materials essential for production, including the movement of materials between different plants.
- Work unit : A work unit is an entity in the factory that adds value to a product. The way a piece of equipment is used determines whether it is classified as a work unit. If the device requires a sequence of steps, or specific settings to carry out its production function, it is a work unit. Work units in ISA-95 are differentiated based on whether they are used in batch or continuous operations, discrete manufacturing, or storage. There are three types of work units:
    - Unit : A unit is a work unit defined for batch and continuous production
    - Work cell : A work cell is a work unit defined for discrete manufacturing
    - Storage unit : A storage unit is a work unit within a storage zone that can be a location and/or a piece of

equipment that is dedicated to the storage of materials and/or equipment

> ### *Extension of ISA-95*

> *The ISA-95 equipment model stops at the work unit level since this is typically the lowest level of equipment that is separately monitored and scheduled in the MES and ERP layers. However, this is not granular enough for us to fully model the factory equipment down to the control layer.*

SA-88 extensions

Often, to model an entire enterprise from the head office down to a single sensor or control input in a factory, the equipment model is extended to include the ISA-88 control and equipment modules:

- Equipment module : An equipment module is part of a larger piece of equipment that can carry out a finite number of specific minor processing activities. It combines all of the necessary physical processing and control equipment to perform these activities. Functionally, the scope of the equipment module is defined by the finite tasks it is designed to carry out. Physically, the equipment module may be made up of control modules and subordinate equipment modules. Equipment modules are typically controlled procedurally through complex logic. They may have multiple states and run through sequences of operations. Equipment modules are connected to other pieces of equipment and systems through control modules.

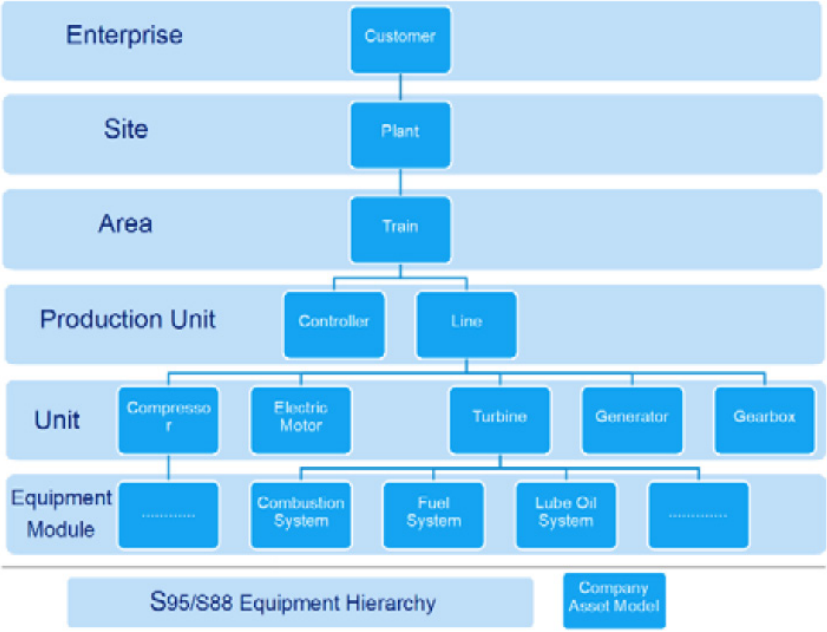
- Control module : A control module is typically a collection of sensors, actuators, other control modules, and associated processing equipment that, from the point of view of control, operate as a single entity. Direct connections to sensors and actuators in the factory are often made in the control module.

To determine the asset model, we need to follow an iterative process that is based on the following steps:

1. Design the conceptual model of the company asset. To do this, you need to identify the physical asset and set up the customer business from a logical perspective. You can then work out which logical resources you need to manage.
2. Fit the designed conceptual model in the hierarchy of the ISA-95 or the ISA-88 equipment model. This means that you need to associate each logical resource identified in the previous step with the corresponding entity in the hierarchy of the equipment model made available by the ISA-95/ISA-88 standards.

In general, the granularity of the model depends on what a company wants to do with the model. A company needs to look at the work processes and the information flow in light of their own business goals.

The following diagram outlines what an asset model could be in an LNG company, including its hierarchy from a logical perspective and the related entities of the ISA-95/ISA-88 standards:



Enterprise
Customer
Site
Plant
Area
Train
Production Unit
Controller
Line
Unit
Compresso r
Electric Motor
Turbine
Generator
Gearbox
Equipment Module
....
Combustion System
Fuel System
Lube Oil System
....
S95/S88 Equipment Hierarchy
Company Asset Model

Figure 3.13 – Asset model example

In this section, we have studied how to organize our components in an asset model. We will find this concept again in *Chapters 7* , *8* , *9* , *10* , *11, and 12* , where we will develop an IIoT architecture.

Summary

In this chapter, we analyzed the main devices involved in the IIoT data flow in an industrial plant in detail. We started with the generation of the data, looking at how physical measurements are captured by the transducers, converted to a digital signal, and passed through the industrial networks by controllers. We learned that industrial networks provide determinism in message delivery, which is a fundamental requisite in the control process. We discovered the enabling mechanisms that can guarantee deterministic networks. We compared a simplified networking model (an EPA) with the full model of the ISO/OSI stack and we looked at the methods that are used to access the physical link of the control network to ensure that the packet is transmitted within a determined time interval. We also learned that there are different types of industrial networks according to the length of the message and the complexity of the industrial device.

After that, we analyzed controllers in detail, looking specifically at microcontrollers, PLCs, and DCSs. We explained their main characteristics, how they differ from one another, and which industrial scenarios they each cover. We learned that they all get their measurements from the sensors as an input, perform control algorithms, and provide the values for the actuators, acting on the physical process as an output.

We then moved our discussion to a higher level, that is, analyzing the SCADA system and Historian. We looked at the main features of a SCADA system and the main mechanisms that allow a Historian to collect, store, and retrieve a huge amount of time series data. We learned a wide range of key terms, including tags and tag names, time series, sampling, polling and unsolicited, store and forward, data filtering, and data compression. Finally, we outlined the ERP system and explained what a MES system is. We also looked at the asset model, as it plays a key role in managing industrial data.

In the next chapter, we will learn more about the OPC and the edge device, looking more closely at how the OPC allows data to pass to the edge. We will then analyze the edge and follow the IIoT data flow until it reaches its destination in the cloud.

4

Implementing the Industrial IoT Data Flow

In the previous chapters, we looked at how a typical industrial scenario works and at the existing technologies that have been in place since 1990. In this chapter, we are going to understand how to connect this old world to the new IoT world. We will look closely at how to connect and gather industrial data from different devices and data sources. In particular, we will explain the Object Linking and Embedding (OLE) for Process Control (OPC) protocol in detail. We'll start by looking at its original implementation in 1996, based on the Microsoft architecture of the Component Object Model (COM) and the Distributed Component

Object Model (DCOM), and track its progress right up to the current Unified Architecture (UA), which is based on open and interoperability standards.

We will then go into detail about some concepts that were outlined in the previous chapters. These include high- and low-frequency data, sampling, data filtering, dead bands, data compression, polling, and unsolicited data collection. We will also introduce other concepts related to the collection of industrial data, such as subscription, collection timeouts, temporary storage, and data buffers.

After that, we will look at edge devices in detail to give the reader a comprehensive idea about their role, functionalities in the IIoT data flow, hardware footprint, software architecture, networking capabilities, and the options that are available in the market. We will also take a look at the emerging models of edge and fog computing.

Following this, we will look at the IIoT data flow from the perspective of a developer. This will include choosing which industrial device to use to gather industrial data, highlighting their strengths and weaknesses; how to collect the data that comes from sensors and actuators through OPC or a fieldbus; how to bubble them to the edge; and, finally, how to transfer them in a reliable way to the cloud.

In this chapter, we will explore the following topics:

- OPC Classic and OPC UA
- The edge, the edge gateway, and fog computing
- Data sources for the IIoT
- Edge deployment and IIoT data flow scenarios

Discovering OPC

No other industrial communications standard has achieved such widespread acceptance across so many different verticals, industries, and equipment manufacturers as OPC Classic. It is used to interconnect a large variety of industrial and business systems.

SCADA, safety instrumented systems (SISs), programmable logic controllers (PLCs), and distributed control systems (DCSs) use OPC to exchange data with each other and with Historian databases, MES, and ERP systems in the corporate world. The reason for the success of OPC Classic is very simple—it is the only truly universal interface that can be used to communicate with different industrial devices and applications, regardless of the manufacturer, software, or protocols used in the control system. This ability has become increasingly important, as the need to exchange information between the different devices of the industrial automation pyramid has increased constantly over the years.

A good example to understand OPC is to imagine it as a language translator. I am Italian, so if I need to talk with a Chinese person, I must learn Chinese or they must learn Italian. Similarly, if I need to communicate with people who speak French, German, Russian, Spanish, or Arabic, I am also obliged to learn these languages. A lot of effort would be required to learn a new language every time I meet someone from a new country, so this is not really a feasible option. However, if everyone decides to adopt and learn a common language, such as English, all I have to do is learn that common language and use that to talk to other people. OPC does exactly this: a common interface, such as OPC, enables communication with various industrial control products, irrespective of the underlying software or hardware. Prior to the introduction of OPC, application developers were required to create unique communication drivers for each control system they intended to connect to. For instance, SCADA vendors needed to produce 10 distinct drivers to interact with the diverse DCSs and PLCs available in the market.

While OPC doesn't eliminate the need for drivers altogether, application vendors no longer need to develop different drivers for each industrial network or processor. The effort required is considerably reduced since each vendor only needs to implement the OPC interfaces. Typically, each manufacturer develops an OPC server for its specific product, using the native protocol for that device, and an OPC client, to interface the OPC servers of the other vendors. In this way, each OPC server will be implemented with full knowledge of the underlying native protocol. Once an OPC server has been established for a particular piece of equipment or application, it significantly simplifies the process of integrating its data into other OPC-compatible software. There are software houses, such as Matrikon, that specialize in this specific area and offer a wide variety of OPC servers for a large number of products.

Initially, the OPC standard was based on the Windows operating system (OS) and Microsoft OPC technology.

These specifications are now known as OPC Classic (http://opcfoundation.org/about/opc-technologies/opc-classic/). In 2006, the OPC Foundation developed the OPC UA (http://opcfoundation.org/about/opc-technologies/opc-ua/) specifications, which use a service-oriented architecture. These implemented the following actions:

- They overcome the limitations that were related to the architecture of OPC Classic
- They address the needs that come from new challenges in security and data modeling
- They provide a more scalable and flexible platform
- They build an independent and open platform that is not linked to a specific technology

The term OPC signifies Open Platform Communications . A significant feature of OPC is that it supports an application programming interface (API), rather than an on-the-wire protocol. OPC supports a higher level of abstraction than communication protocols such as Ethernet, TCP/IP, or fieldbus protocols, such as EtherNet/IP or PROFINET. This is true both for OPC Classic and OPC UA. We will analyze both in detail in the following sections.

OPC Classic

In 1995, various companies decided to create a working group to define an interoperability standard (https://opcfoundation.org/about/opc-foundation/history/). These companies were the following:

- Fisher Rosemount Intellution
- Intuitive Technology Opto22
- Rockwell
- Siemens AG

Members of Microsoft were also invited to provide the support necessary. The goal of the working group was to define a standard for accessing information in the Windows environment, based on the current technology at that time.

The technology that was developed was called OPC. In August of 1996, the first version of OPC was defined (https://opcfoundation.org/about/opc-foundation/history/). The following diagram shows the different layers of OPC Classic with the underlying critical communication protocols—COM, DCOM, and remote procedure call (RPC):

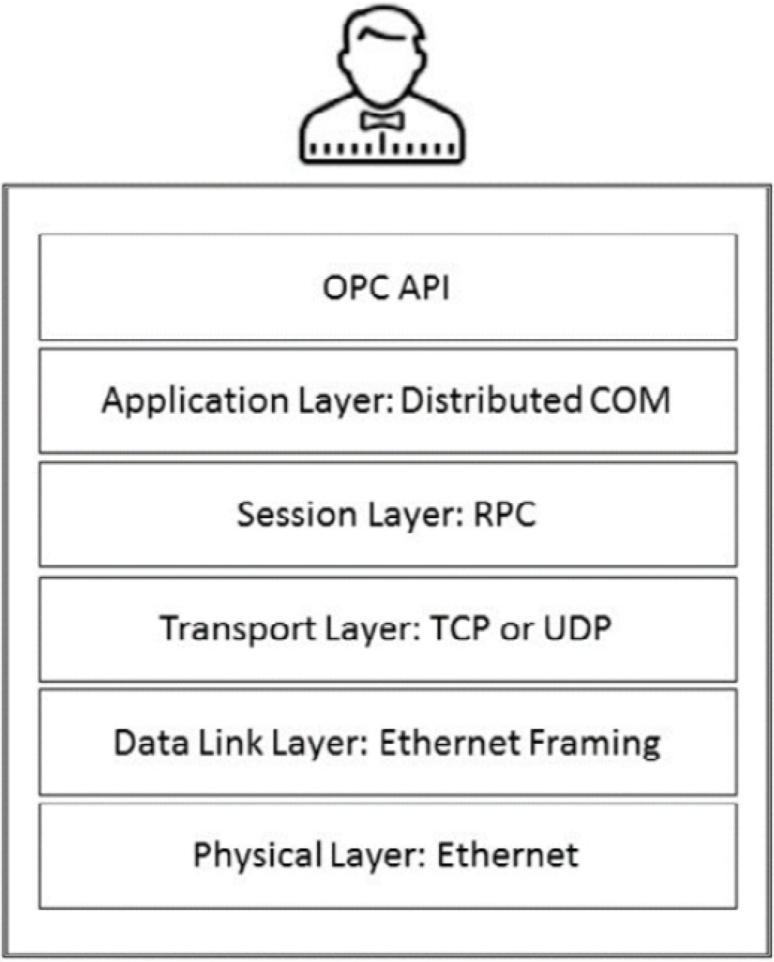

OPC API
Application Layer: Distributed COM
Session Layer: RPC
Transport Layer: TCP or UDP
Data Link Layer: Ethernet Framing
Physical Layer: Ethernet

Figure 4.1 – OPC layering

COM, a software architecture created by Microsoft, facilitates the development of component-based applications by encapsulating reusable code components. This design permits other applications to access and utilize these encapsulated components without being concerned about their underlying implementation. COM objects can be substituted with newer versions without necessitating application rewrites.

DCOMs are network-aware versions of COMs. A DCOM tries to hide the differences between the COM objects that are running on the same computer and the COM objects that are running remotely on a different computer from the software developers. To achieve this, all the parameters must be passed by value. This means that when invoking a function exposed by a COM object, the caller must pass the related parameters by value. The COM object, on the other hand, will reply to the caller, passing the results by value. The conversion of parameters into transferable data format is referred to as marshaling . Following marshaling, the data stream undergoes serialization, transmission, and deserialization on the receiving end to restore the original data order.

COM components on the same network can exchange information through DCOM's use of RPCs, which facilitates the process in a transparent manner.

Microsoft's RPC mechanism enables developers to request the execution of remote programs without the need for creating custom server procedures.

The client program transmits the required inputs to the server, which then runs the corresponding program and sends back the resulting output as a message to the client.

OPC Classic had several limitations:

- The standard was coupled to a specific technology. OPC Classic, in fact, was built around and on top of Microsoft technology.
- COM traffic relies on several undetermined network ports to be opened, so it is often blocked by firewalls.
- DCOMs and RPCs were weighty and complicated mechanisms. DCOM objects often suffered from a lack of performance and were difficult to maintain.

The following diagram shows the implementation of the same data retrieval scenario using two different architectures and two different kinds of software:

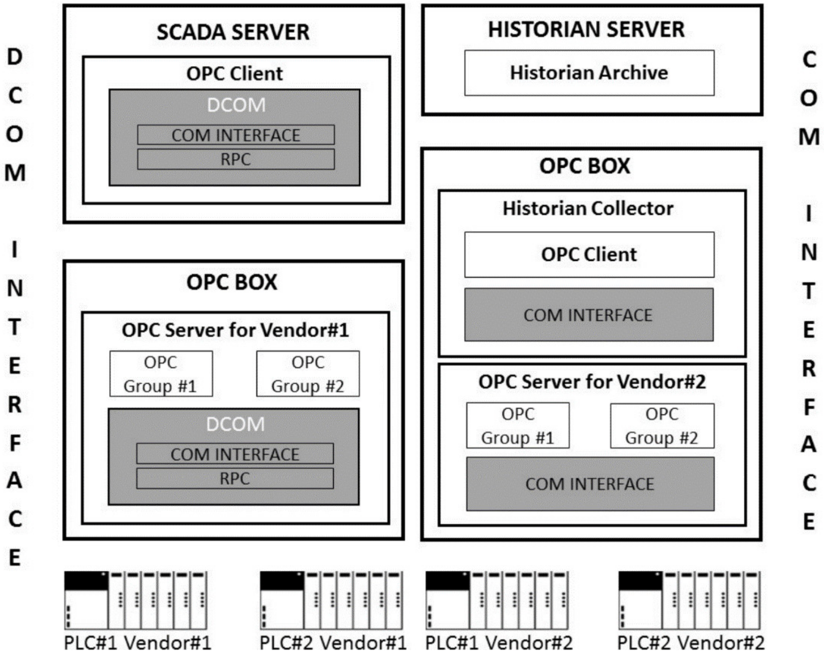

DCOM INTERFACE
SCADA SERVER
OPC Client
DCOM
COM INTERFACE
RPC
OPC BOX
OPC Server for Vendor#1
OPC Group #1
OPC Group #2
DCOM
COM INTERFACE
RPC
HISTORIAN SERVER
Historian Archive
OPC BOX
Historian Collector
OPC Client
COM INTERFACE
OPC Server for Vendor#2
OPC Group #1
OPC Group #2
COM INTERFACE
COM INTERFACE
PLC#1 Vendor#1
PLC#2 Vendor#1
PLC#1 Vendor#2
PLC#2 Vendor#2

Figure 4.2 – OPC layering

We have two PLCs connecting to a computer that is running the OPC Classic server for the PLC vendor. In the scenario on the left, named DCOM INTERFACE, the PLCs and the OPC server communicate using the native PLC protocol. The OPC client running on the SCADA computer accesses the data in the OPC server through the

DCOM since they are running on different computers.

In the scenario on the right, named COM INTERFACE, the PLCs and the OPC server communicate using the native PLC protocol, but the OPC client is embedded in a Historian agent running on the same machine as the OPC server and therefore accesses the data in the OPC server through COM. The data gathered by the Historian agent through the OPC client is then sent to the Historian server through TCP/IP. In the scenario on the right, the communication between the OPC client and the OPC server is done using COM since they are running on the same Windows box, therefore avoiding the performance and security issues that affect DCOM communication.

Let's learn about the data model and retrieving data in OPC Classic.

The aims of the OPC Classic standard are as follows:

- To structure the data on the server side in order to simplify the data gathering on the client side
- To define the communication services and the standard mechanisms to exchange data

Basically, the OPC Classic standard works as follows:

- The server manages all available data.
- The server sends queries for the data coming from the devices on demand and updates an internal cache on a regular basis. The server initializes and manages a cache for each group of variables requested by an OPC client. The scan rate related to an OPC client group cannot be less than the scan rate used by the OPC server to gather the data from its devices to update its internal cache. It is a good idea to set the OPC client to carry out cache reads and set them to refresh at double the rate that the OPC server scans the devices.
- Each piece of exchanged data has its value signed by its timestamp and marked with a quality.
- Exchanging the data includes reading it, writing it, and automatically updating it when values change. The reading, or polling, is carried out by the OPC client, which sends queries for the data of a group on a regular basis. The writing stage can either be synchronous or asynchronous. The automatic updates, or the unsolicited data collection, use a request update rate that is provided by the OPC client. The OPC server for each update checks whether the absolute value of the cached value minus the current value is greater than the dead band specified by the client multiplied by the range that is

configured for that variable. This can be written as follows:

```
IF absolute value of (last cached
value -current value) >
(PercentDeadband/100) * Range
THEN cache is updated, and the client
is notified through a callback
function
```

The OPC server categorizes information into distinct groups. The grouping of related items enhances the server's overall efficiency. Servers can contain one or more groups of items. There are two different types of groups:

- Public groups : Available for any client to access them
- Local groups : Only accessible to the client that created them

Let's take a look at an example. In an OPC server that is connected to a MODBUS-capable PLC, we might configure a water-level group on the SCADA with the following five members:

- Setpoint (SP)
- Control output (CO)
- Process variable (PV)

- Low-water alarm (`LoAlarm`)
- High-water alarm (`HiAlarm`)

The SCADA system is capable of registering a water-level group consisting of five items. It can read the current values for each item either at set intervals or upon detection of a change.

The HMI could also have write access to the SP variable to set new values.

A significant advantage of OPC is that you do not have to deal with the control device's internal architecture directly. The software can work with named items and groups of items, simplifying the access, instead of dealing with raw register numbers and data types.

This also makes it easier to add or change control systems, when migrating from a proprietary protocol to an Ethernet-based protocol, for example, without altering the client applications. The following points provide some suggestions related to how to implement a data flow based on OPC Classic:

- If possible, run the OPC Data Access (DA) client on the same Windows box as the OPC DA server to avoid the configuration and performance issues that are related to DCOM.
- The OPC DA client can ask the OPC DA server for readings from either the device or the cache. Requests for readings directly from the device instead of the OPC cache should be limited to specific use cases where you need to have the current values related to very few signals. If you need to read a large number of tags on a regular basis, it is better to use cache readings.
- To avoid overloading the communication channel and the OPC server, group the tags according to the dynamic of their related signals, setting a different polling time for each group. For instance, you could classify the signals according to their behavior in high, medium, and low dynamic signals and assign a different polling time to each of them. You might assign 5 seconds for the high dynamic signals, 30 seconds for the medium dynamic signals, and 60 seconds for the low dynamic signals. The OPC server will then create three different internal caches and refresh them accordingly. Putting all the tags together and configuring just one group for all of them would force you to assign it a lower polling time of five seconds. The OPC server would have to refresh all tags every five seconds, even if many tags would not change meaningfully in that time interval.
- Configuring the OPC group to have unsolicited data readings would force the OPC server to invoke a callback if any of the tags configured in a group change. The callback mechanism is not very efficient,

especially when used with DCOM, where the OPC DA client does not run in the same Windows box as the OPC DA server. For this reason, it should be limited to situations in which you are interested in only a few signals with a low dynamic that don't change their values very frequently.

OPC UA

The first response of the OPC Foundation to the rising limitations related to the adoption of the COM and DCOM architecture was the development of OPC XML DA . This kept the fundamental characteristics of OPC but adopted a communication infrastructure that is not linked either to a manufacturer or to a specific software platform. The conversion of the OPC DA specifications into versions based on web services was not enough to fulfill the needs of factories that increasingly interact and integrate with the corporate and external world.

The OPC UA was therefore developed with the aim to replace all existing COM-based versions and overcome security and performance issues, satisfying the need for platform-independent interfaces and allowing the creation of rich and extensible data models to describe complex systems without losing functionality. OPC UA is based on a service-oriented approach defined by the IEC 62451 standard. It has the following aims:

- To use OPC components on non-Windows platforms
- To make it possible to embed its core components in small devices
- To implement standard communication through firewall-based systems

From a technical perspective, OPC UA works as follows:

- The API isolates the client and server code from the OPC UA stack
- The UA stack converts the API calls into messages
- The UA stack receives messages by sending them to the client or the server through the API

This is shown in the following diagram:

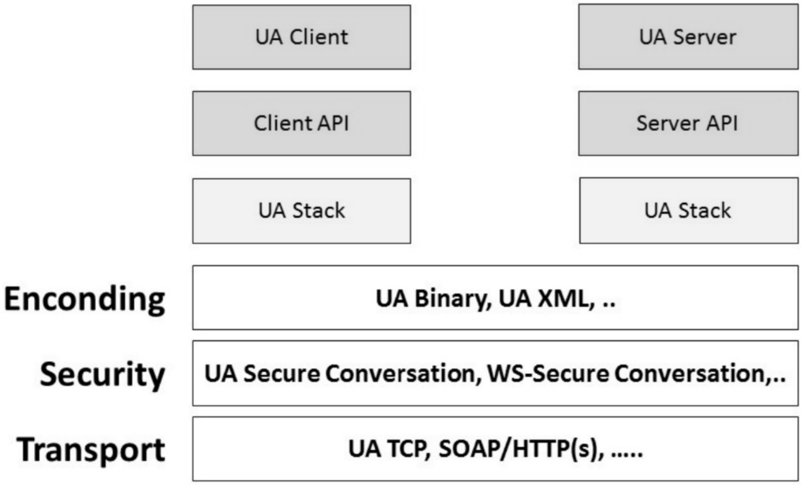

UA Client
UA Server
Client API
Server API
UA Stack
UA Stack
Enconding
UA Binary, UA XML, ..
Security
UA Secure Conversation, WS-Secure Conversation,..
Transport
UA TCP, SOAP/HTTP(s), ...

Figure 4.3 – OPC UA stack

The OPC UA information model

The information model is the conceptual representation of the objects that you manage to carry out your business. The same pieces of physical equipment and the associated pieces of information can be modeled in different ways, according to the requirements of the specific process and environment.

In OPC Classic, it was only possible to manage simple data; the coding of any semantics was not possible, since the information model was flat. For example, in the case of a temperature measured by a sensor, the only information that would have been available to understand the meaning of that data was the name of the variable and a few other elements, such as the unit of measurement. The basic specifications of OPC UA instead provide an infrastructure to structure the information model according to your needs. It also provides a built-in, ready-to-use information model, which gives manufacturers the freedom to create their own extensions according to their devices. This means that clients can access this information regardless of the manufacturer since it will be made available through the same basic model. The basic principles of information modeling in OPC UA are as follows:

- Use of object-oriented techniques, including hierarchies and inheritance
- The same mechanism is used to access the types and instances
- Information is made available by exploiting fully-connected node networks

- The hierarchies of the data types and the links between the nodes are extendable
- There are no limitations on how to model information
- Information modeling is always placed on the server side

The set of objects and related information that an OPC UA server makes available to the clients is the address space . You can think of the address space as similar to the implementation of the OPC UA information model. The address space of the OPC UA is a set of nodes that are connected by references. Each node has properties, which are called attributes . A specific set of attributes must exist in all nodes. The relationships between nodes, attributes, and references are shown in the following diagram:

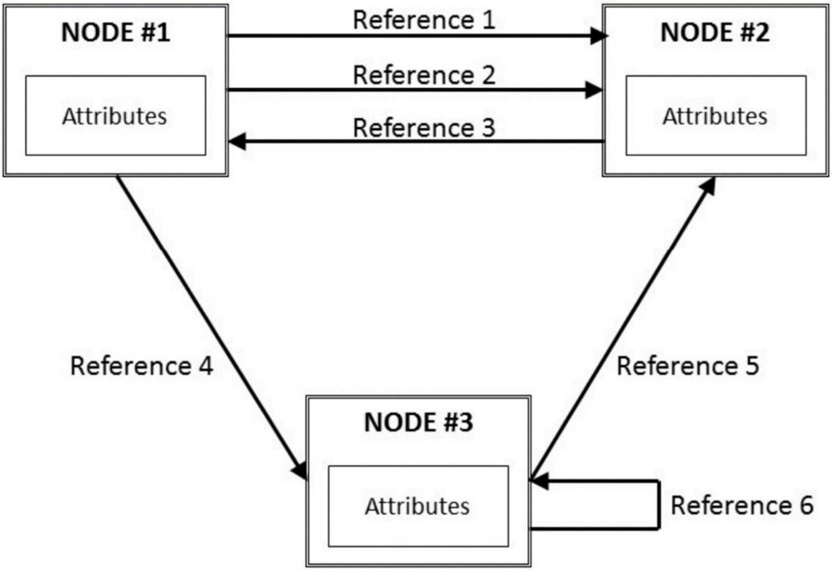

NODE #1
Attributes
Reference 1
Reference 2
Reference 3
NODE #2
Attributes
Reference 4
Reference 5
NODE #3
Attributes
Reference 6

Figure 4.4 – OPC UA information model

Nodes can belong to different node classes, depending on their specific goal. Some nodes might represent instances, others might represent types, and so on. In the OPC UA, there are eight standard node classes: variable, object, method, view, data type, variable type, object type, and reference type. In OPC UA, the most commonly used node classes are *object* , *variable* and *method* Here is some more information about the node classes:

- The object node class structures the address space and can be used to group variables, methods, or other objects.

- The variable node class represents a value. Clients can read, write, or subscribe to it.
- The method node class represents methods that have input and output parameters that clients can invoke through the call service to get a result. They are designed to work quickly. Some of the quick actions that can be managed through methods include opening a valve, starting a motor, or even calculating the results of a simulation based on the input parameters provided.
- A reference describes the relationship between two nodes and is uniquely identified by the source node, the target node, the reference type, and its direction. We can think of references as pointers to other nodes. They are not accessible directly but can be accessed by browsing the nodes. The reference type is the way in which the link between the connected nodes is modeled. OPC UA defines a set of built-in reference types organized hierarchically, but each server has the option to define its own specific set by extending the basic one. The reference types are exposed in the address space as nodes and this allows clients to easily recover information on the references that are used by the OPC UA server in which the client is interested.
- A view is used to restrict the number of nodes and references that are visible in an extended address space. Using views, servers can provide different representations of their address space to different clients that they are connecting to, depending on their use cases. There are two ways to use views in OPC UA:
    - A view can be represented in the address space as a node that provides an entry point to the data to be displayed. All the nodes that are part of a view are accessible, starting from the starting node.

    - The node ID of the view node can be used as a filtering parameter for address space navigation. In this way, the servers can hide the references to the nodes that are not part of the view and the clients only see a subset of the address space.

> ### *OPC UA Semantic*

> *The OPC UA protocol is also utilized in robotics due to its capability to semantically describe the structure of plants or assets. This allows many vendors to automatically read the robot's structure and import its data seamlessly, eliminating the need for resource-intensive mapping (see also https://opcfoundation.org/markets-collaboration/robotics/).*

OPC UA sessions

OPC UA provides a client-server communication model that includes status information. This status information is associated with a session. A session is defined as a logical connection between a client and a server. Each session is independent of the underlying communication protocol; any issues occurring at the protocol level do not automatically cause the termination of a session. The sessions terminate following an express request from the client or due to the client's inactivity. The inactivity intervals are set during the creation of a session.

The OPC UA security model

The OPC UA security model is implemented through the definition of a secure channel, on which a session is based. A secure channel makes the exchange of data secure in different ways:

- It ensures the integrity of the data using digital signatures. It ensures confidentiality through encryption.
- It carries out the authentication and authorization of applications using X.509 certificates.

The entities involved, as shown in the following diagram, are the Application Layer, the Session, and the Transport Layer:

- The Application Layer is used to transmit information between clients and servers that have established an OPC UA session
- An OPC UA session is established on a secure channel (located in the Communication Layer), which makes it safe to exchange data

The Transport Layer is the level responsible for transmitting and receiving data through a socket connection, to which error-handling mechanisms are applied to ensure the system is protected against attacks, such as denial of service (DoS).

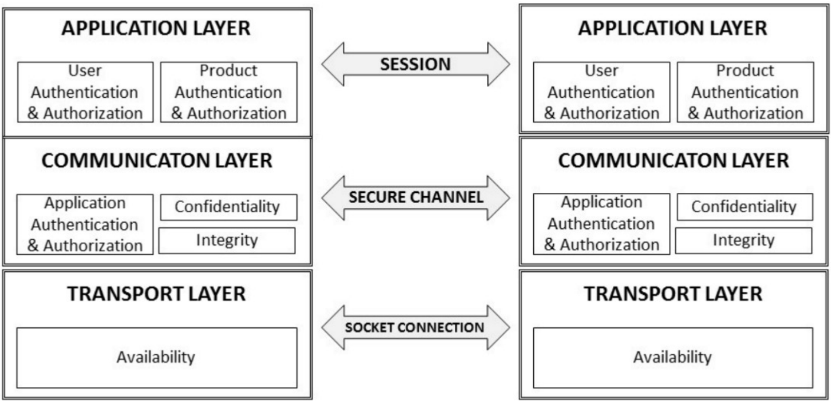

APPLICATION LAYER
User Authentication & Authorization
Product Authentication & Authorization
SESSION
APPLICATION LAYER
User Authentication & Authorization
Product Authentication & Authorization
COMMUNICATON LAYER
Application Authentication & Authorization
Confidentiality
Integrity
SECURE CHANNEL
COMMUNICATON LAYER
Application Authentication & Authorization
Confidentiality
Integrity
TRANSPORT LAYER
Availability
SOCKET CONNECTION
TRANSPORT LAYER
Availability

Figure 4.5 – OPC security layer

The creation of a secure channel is based on the endpoint and each server offers one or more endpoints. Each endpoint has the following features:

- Endpoint URL : The endpoint's network address is utilized by the client to establish a secure connection.
- Server Application Instance Certificate : This is the public key of the server used by the client to make the exchange of data secure.
- Security policy : This is the set of algorithms used in security

mechanisms and also includes the length of the key that is used. An example of a security policy is the Advanced Encryption Standard (AES) with a 128-bit key.

- Security mode : This ensures the authentication at the level of the application. There are three different modes that can be used: **`SignAndEncrypt`** , **`Sign`** , or **`None`** .
- Authentication : This refers to the mechanisms used to authenticate a user during the creation of a session by means of a username and password, a certificate, or through anonymous authentication.
- Transport protocol : This specifies the network protocol used.

The creation of a secure channel is based on three discovery services: **`FindServers`** , **`RegisterServer`** , and **`GetEndpoints`** . We also have a discovery server, where all the active OPC UA servers (**`RegisterServer`**) are registered. It is the client's responsibility to ask the discovery server for a list of the active OPC UA servers (using **`FindServers`**) and then to request the chosen server and the list of available endpoints (using **`GetEndpoints`**). To establish a connection between an OPC UA client and an OPC UA server, we have to follow four steps. The first two are shown in the following diagram:

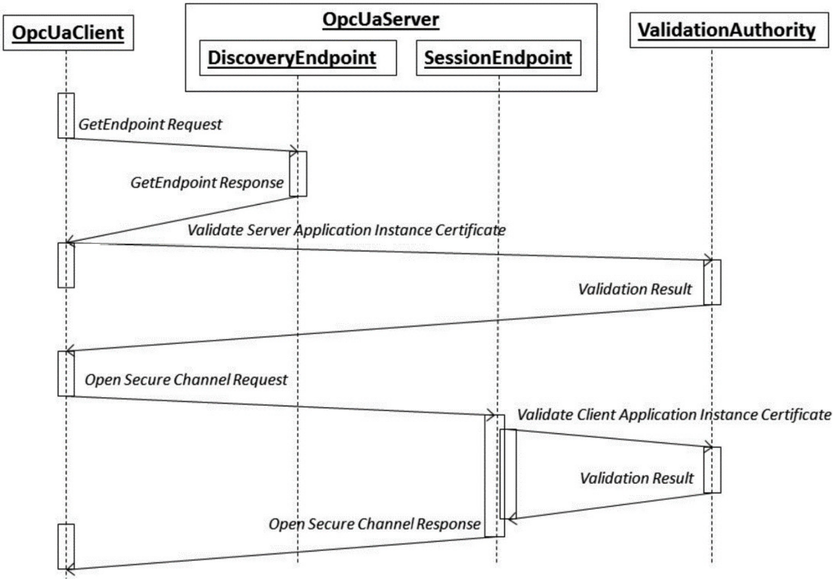

OpcUaClient
OpcUaServer
DiscoveryEndpoint
SessionEndpoint
ValidationAuthority
GetEndpoint Request
GetEndpoint Response
Validate Server Application Instance Certificate
Validation Result
Open Secure Channel Request
Validate Client Application Instance Certificate
Validation Result
Open Secure Channel Response

Figure 4.6 – OPC security sequence diagram, Steps 1 and 2

The details of the four steps are as follows:

1. Set the configuration options for the secure connection. If the application is pre-configured and/or it knows how to connect to the server, we can skip this step. If not, the client must send a `GetEndpoints` request to the discovery endpoint of the server to which it wants to connect to receive the descriptions of the existing session endpoint and the related available security configurations, including security policies, security modes, and server Application Instance Certificate.
    - The security policy defines the algorithms to be used for

signing and encrypting messages, while the security mode defines the type of security.

    - The client selects a session endpoint with a supported security configuration and validates the server Application Instance Certificate. This is done by checking its validity status with the associated validation authority (VA).

4. If the certificate is reliable, we send an Open Secure Channel request in line with the security policy and the security mode of the selected session endpoint:

    - If the security mode is `None` , the Open Secure Channel request is sent without any security mechanisms.
    - If the security mode is `Sign` , the Open Secure Channel request is sent using the private key of the client Application Instance Certificate as a signature.
    - If the security mode is `SignAndEncrypt` , the Open Secure Channel request is sent after encrypting it using the public key of the server Application Instance Certificate.
    - The security policy specifies which encoding and signature algorithms should be used for signing and encrypting messages.
    - Once the server receives the Open Secure Channel message, the server validates the client's Application Instance Certificate by a request to the VA.

- If the certificate is considered valid, the message is interpreted according to the security policy and the security mode. The message is decoded using the server's private key and the signature is verified using the client's public key.
- The server sends the response to the client in the same way as the client sent the request.
- The secure channel is established.
- The creation of the secure channel is mainly to create symmetric keys, which enable the exchange of secret information between clients and servers. The symmetric keys mean we can avoid using public key cryptography (asymmetric keys), which is less efficient in terms of computational speed.

14. Establish a new session on the already-secure communication channel:

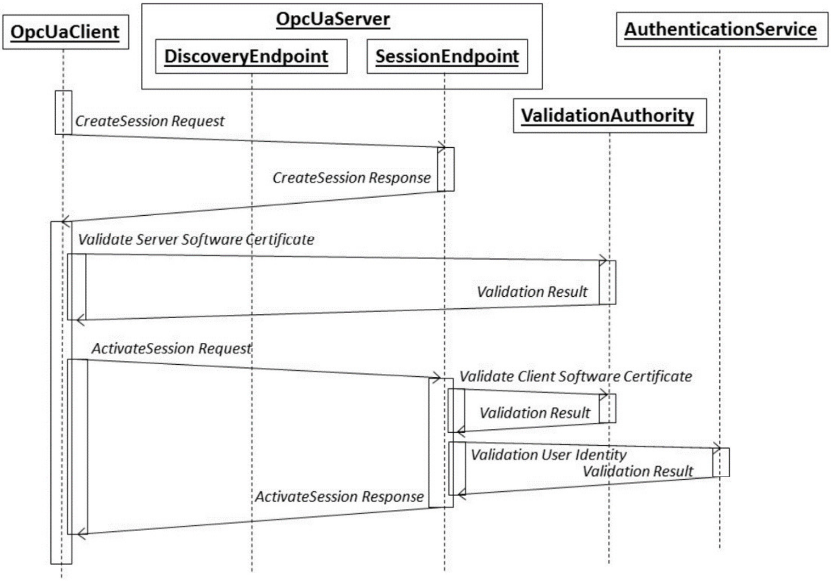

OpcUaClient
OpcUaServer
DiscoveryEndpoint
SessionEndpoint
AuthenticationService
ValidationAuthority
CreateSession Request
CreateSession Response
Validate Server Software Certificate
Validation Result
ActivateSession Request
Validate Client Software Certificate
Validation Result
Validation User Identity
Validation Result
ActivateSession Response

Figure 4.7 – OPC security sequence diagram, Steps 3 and 4

A `CreateSession` request is sent to the server. The server replies and provides its software certificates to communicate its functionality and to demonstrate the ownership of the certificate used in the creation of the underlying secure channel.

1. Activate the session that was just created. The customer sends an `ActivateSession` request to the server, with the user's credentials and the client's software certificates. The credentials can either be represented by an X.509 certificate that has been validated by a VA, or by a username and password pair. Once the user credentials and the software certificates have been validated by the server, the

session is established and active, and the client can start accessing the server data.

The OPC UA data exchange

The easiest way to exchange data between an OPC UA client and a server is to use read and write services. The read and write services are optimized to transfer a bunch of data rather than a single piece of data or a few values. They allow us to read and write either values or attributes of nodes.

The read service has the following parameters:

- `maxAge` : This is the maximum time taken for the values to be returned. It is specified by the client. It forces the server to access the device (for example, a sensor) if the copy in its cache is older than the `maxAge` parameter configured by the client. If `maxAge` is set to zero, the server must supply the current value by always reading it directly from the device.
- Type of timestamps : In OPC UA, two timestamps are defined: the source timestamp and the server timestamp. The source timestamp is the timestamp that comes from the device, while the server timestamp is the timestamp that comes from the OS where the OPC UA server is running.

- List of nodes and attributes : The list of nodes and attributes are as follows:
    - **`NodeId`**
    - **`AttributeId`** for instance value
    - **`DataEncoding`** : This allows the client to choose the appropriate data encoding and the default values are XML and UA binary

The parameters of the write services are the following:

- List of nodes, attributes, and values : The list of nodes, attributes and their values are as follows:
    - **`NodeId`**
    - **`AttributeId`**
    - Value to write
    - **`Source timestamp`** : Null if not set
    - **`Timestamp server`** : Null if not set

A client can create one or more subscriptions for each session. For each subscription, the client can create monitored items, which are related to the real items. Each monitored item can create a notification, which can either be linked to the changes in the values of the attributes or variables selected by the client, or it can be connected to when events occur.

Each monitored item continuously produces notifications either until the subscription is cancelled, or the monitored items are deleted. Clients can use subscriptions to receive updates on a regular basis.

The following diagram represents the relationships between sessions, subscriptions, and monitored items:

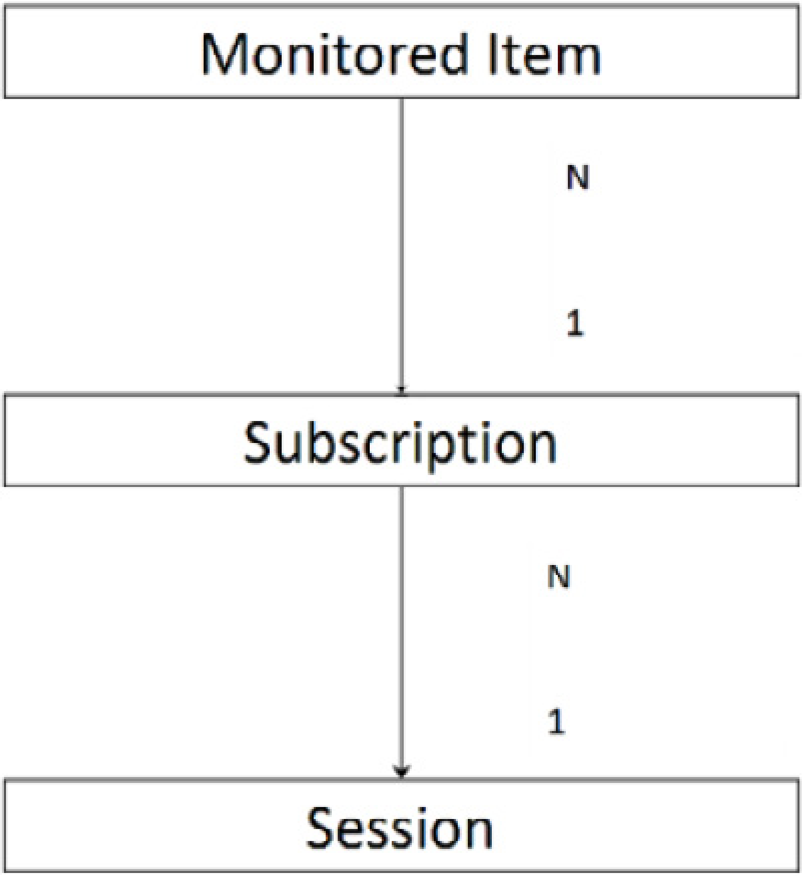

Monitored Item
N
1
Subscription
N
1
Session

Figure 4.8 – OPC UA subscriptions

The subscription collects and groups all notifications related to it on a regular basis. All notifications thus grouped are inserted in a NotificationMessage , which is to be sent to the client. Monitored items cannot exist if a subscription is not created by the client within a session; the client must create a subscription before the monitored item can be defined. A subscription has several parameters. The main ones are as follows:

- `PublishingInterval` : This defines the time interval used by the server to create a *NotificationMessage* for the given subscription
- `PublishingEnabled` : This is a Boolean value that enables the forwarding of the *NotificationMessage* to the client

In addition, each subscription manages the following:

- A keep-alive counter that counts the number of publishing intervals that have elapsed with no notifications sent to the client
- A lifetime counter that counts the consecutive number of publishing intervals that have elapsed with no actions carried out by the client
- A queue of the *NotificationMessage* sent and not acknowledged by the client

A subscription allows us to create an automatic mechanism to write messages to be sent to the client, which is called *NotificationMessage* . This contains the values produced by the monitored items that are created for that subscription.

Monitored items are defined within a subscription to monitor server items. These can be one of the following:

- Attributes : Any attribute of any node can be associated with a monitored item. Attributes are only monitored to indicate when their value changes. A value change produces a notification.
- Variable : The value attribute of the nodes belonging to a `NodeClass` variable can be associated with a monitored item. The value attribute of a variable is monitored to check for changes in its value or status under specific conditions. If the conditions are met, the related change triggers a notification by the monitored item.
- Objects and view : The nodes belonging to the node class's object and view can be associated with a monitored item. These are monitored to detect the occurrence of a particular event.

For every change of attribute, value, or status, or for each event monitored by the monitored item, a notification is produced. Several notifications are packaged by the subscription within a `NotificationMessage` at time intervals according to the publishing interval.

All monitored items have some common settings. The most important of these is `SamplingInterval`

. This defines the frequency in milliseconds at which the server samples the items with which the monitored item is associated. The default value of the sampling interval is the same as the publishing interval of the related subscription. A client must set the sampling interval to zero (because doesn't make sense for events) if the subscription is related to events. If the client specifies a sampling interval that is not supported by the server, the server assigns the most appropriate range to the monitored item. The sampling performed by the OPC UA server and the scan cycle used to get the data from the underlying device are usually not synchronized. This can cause delays in detecting changes.

OPC UA notifications

As mentioned in the previous section, the client can define one or more subscriptions within an active session. For each subscription, it can create monitored items, which produce notifications that are stored in specific queues. According to the frequency specified by the publish interval, for every subscription, the current contents of all queues of the monitored items related to the subscription are merged into **`NotificationMessage`** , which is to be delivered to the client. In OPC Classic, this information was sent through callbacks, which allowed the server to invoke methods on the client. In OPC UA, the communication mechanism is based on unidirectional connections that don't use the callback mechanism. This means it can easily be managed by firewalls and is independent of the underlying transport layer.

Through the Publish Service request, the client sends a request to the server, expecting to receive a Publish Service response that contains a **`NotificationMessage`** within the expiration time specified by the publishing interval. A Publish Service request does not refer to a specific subscription, but just to the session in which the request has been submitted. So, for each Publish Service request, the server will be required to forward to the client a `NotificationMessage` from those produced by the subscriptions of the active session. If several `NotificationMessage` have been produced and are ready to be transmitted, the server will decide which `NotificationMessage` to send to the client, either according to the priority of the subscription or by using a round-robin algorithm.

The following diagram shows the notification mechanism we just described. For each Publish request received, the server selects a subscription, picks a `NotificationMessage` , and is sent a Publish Response. The following diagram also shows a Publish request queue, in which all requests that are advanced by the client and received in the session are queued:

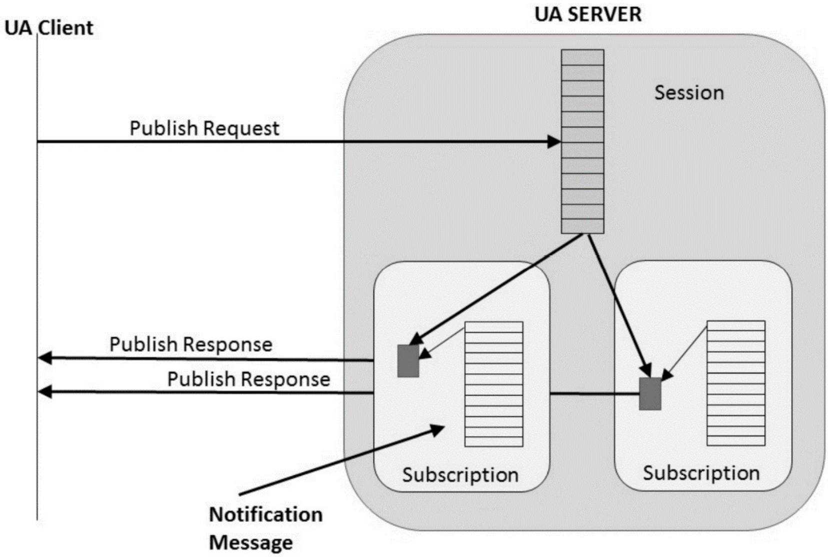

UA Client
UA SERVER
Session
Publish Request
Publish Response
Publish Response
Subscription
Subscription
Notification
Message

Figure 4.9 – OPC UA notification mechanism

We have seen how to work with the OPC protocol; in the next section, we will study the edge component: the edge is the *bridge* that connects the industrial and SCADA world to the cloud world.

Understanding the IIoT edge

The edge is the junction point between the industrial world and the IoT world in the cloud. The edge can be decomposed into three macro components, as shown in the following diagram:

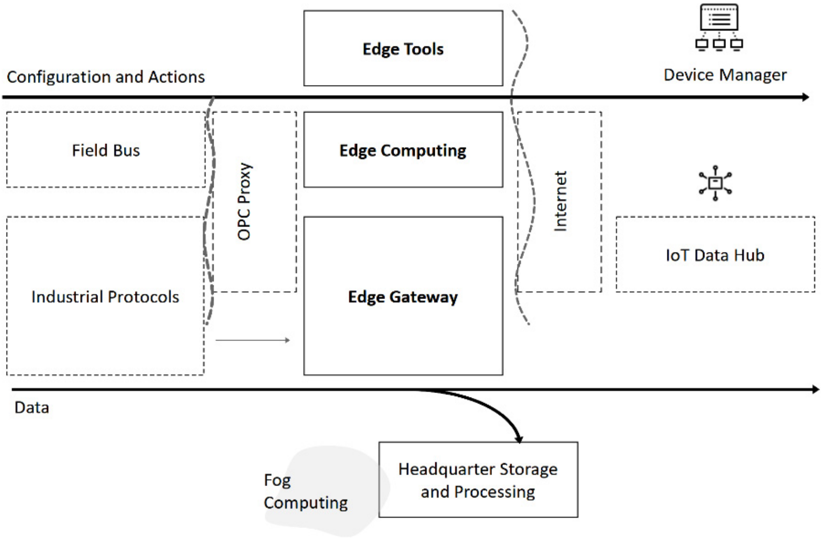

Configuration and Actions
Edge Tools
Device Manager
Field Bus
OPC Proxy
Edge Computing
Internet
Industrial Protocols
Edge Gateway
IoT Data Hub
Data
Fog
Computing
Headquarter Storage
and Processing

Figure 4.10 – The macro components of the edge

The edge gateway is the core and is responsible for forwarding the data from the site to the IoT data hub, whether this is on the cloud or not. The edge tools are utilities for configuration, log management, and patching the edge OS, either from a remote or a local user interface. Edge computing is a new, upcoming component that uses data to perform an action at the site level or to provide insights to headquarters.

So far, the edge has been limited to collecting and forwarding data using the edge gateway to the IIoT middleware, whether this is on the cloud or not. Recently, however, industrial companies have been able to turn data into actionable intelligence using edge computing, which is available at the edge side. In 2017, Gartner declared the following:

The Edge will eat the cloud.

While this statement might seem a bit controversial, it highlights the role that the edge has played over the past two years. Industrial companies, after an initial phase of absolute cloud-centricity, have realized that it is not always possible to do everything in a remote location. The reasons behind this are as follows:

- Data export : National regulations could prohibit the exportation of sensitive or high-risk data. For instance, Saudi Arabia applies very strict controls on exporting data related to oil and gas. China does not allow the exportation of data at all.
- Network bandwidth : The network bandwidth may not be sufficient to transmit all kinds of industrial data. For instance, high-frequency data, such as the vibrations related to a piece of rotating equipment, needs to be collected on a frequency in a range of between 1 and 50 KHz.
- Network latency : Advanced process controls or analytics related to changes in data for profiling asset behavior in a small time window suffer from high and variable network latency. In general, asset optimizations are required to carry out corrective actions as fast as possible within a deterministic time interval.
- Data connectivity : Process optimizations or asset maintenance are required to carry out actions or trigger the company workflow for the

replacement of components without an internet connection.

The IoT edge versus the IIoT edge

There are a few fundamental differences between the IoT edge and the IIoT edge. From the perspective of data collection, the IoT edge manages from 10 to 50 measures per piece of equipment, whereas the IIoT edge can manage from 300 to 30,000 measures, gathering data from multiple assets scattered across the entire production plant. The IIoT edge should also provide a simple way to configure measures from remote sites through the Edge Manager. Updating the signals to be monitored is a weekly activity that should be performed by the operations manager. Moreover, industrial security requires a strong cybersecurity assessment and a high-fidelity architecture because we seldom connect the sensors over the internet directly. Fortunately, the IIoT edge is less restrictive in terms of budgeting. For example, when monitoring a fridge or a washer, we won't spend more than $50, whereas monitoring a $1,000,000 turbine would require a lot more money.

From the perspective of the computational architecture on the edge side, the IoT has introduced the concept of the fog.

The fog versus the IIoT edge

A few years ago, Cisco coined the term fog computing to describe a layer of computing at the edge side. This layer allows us to preprocess data to be transported to the IoT data hub quickly and securely. Fog computing pushes the elaboration and computation of resources to the local network and processes data in a fog node or IoT gateway. In the industrial sector, the edge has the ability to work in collaboration with a local headquarters, where it can find more computational resources before sending the data to the centralized IoT data hub, whether this is in the cloud or not.

The edge architecture

In the previous section, we identified the three macro components of an edge device to understand their features. In this section, we are going to analyze the three macro-components from the architectural perspective by identifying their components, which are shown in the following diagram:

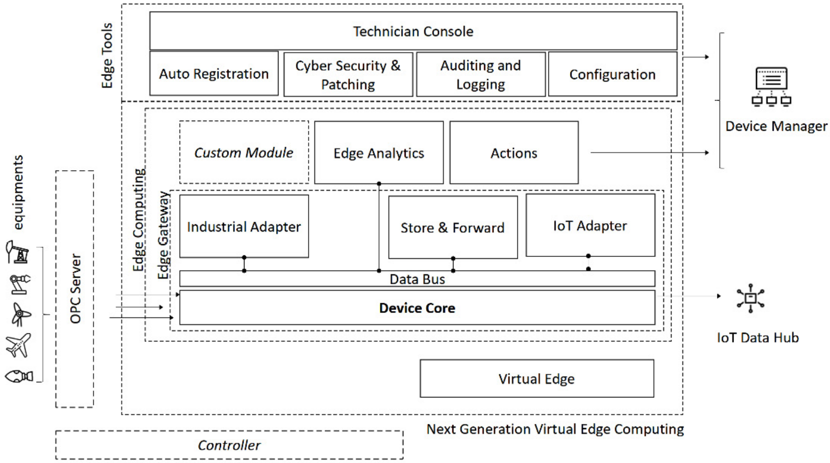

Edge Tools
Technician Console
Auto Registration
Cyber Security & Patching
Auditing and Logging
Configuration
Device Manager
equipments
OPC Server
Edge Computing
Edge Gateway
Custom Module
Edge Analytics
Actions
Industrial Adapter
Store & Forward
IoT Adapter
Data Bus
Device Core
IoT Data Hub
Virtual Edge
Next Generation Virtual Edge Computing
Controller

Figure 4.11 – The IIoT edge architecture

The edge gateway

The edge gateway is the core of the edge device. The main responsibility of the edge gateway is to connect to the industrial data source to gather the data and send it to the IIoT data hub through an internet protocol, such as Message Queue Telemetry Transport (MQTT), Constrained Application Protocol (CoAP), HTTPS, or Advanced Message Queuing Protocol (AMQP).

The most important components of the edge gateway are the industrial adapter and the IoT adapter . The industrial adapter usually subscribes to the data from the industrial field and publishes it on a data bus. Typically, it implements the connector or connectors for the selected device, acting as a data source in our IIoT data flow and making them available on the edge data bus. The IoT adapter, on the other hand, grabs the industrial data from the data bus and forwards it to the IoT data hub. An important part of the edge gateway is the store-and-forward component. This is a general mechanism for storing the data in temporary local storage to make the data transfer resilient to the instability of the network. We discussed this previously when analyzing Historians.

In the edge gateway, the store-and-forward component is crucial since the data transfer occurs over the internet in a wide area network (WAN) rather than in a local area network (LAN). In a WAN, the instability and the latency of the connectivity channel are very high. The store-and-forward mechanism can either offer a limited memory buffer that covers a short period of connectivity downtime, or a large and specific storage area on a disk drive that can cover a long period of connectivity downtime or a huge flow of data.

The range of the time window for which the data transfer must be ensured depends on the specific scenarios and the physical resources of the edge memory and storage.

An edge gateway must have the following features:

- The ability to gather data from different industrial data sources through OPC or by implementing interfaces that are directly connected to the fieldbus protocols (such as MODBUS, PROFINET, or EtherNet/IP).
- The ability to establish a secure channel with the IIoT data hub and to manage certificates and authorization.
- The ability to send data to the IIoT data hub and, in some situations, to compress it.
- Implements a store-and-forward mechanism so that it is resilient against the instability of the internet connection.
- Provides the option to extend its features by means of plugins. An example could be the implementation of a connector for a specific industrial protocol.

The edge computing

Edge computing is a relatively new concept. It is made up of analytics running on the edge for filtering, aggregation, and advanced anomaly detection. These analytics should be deployed remotely from the cloud through the Edge Manager. The Edge Manager running on the cloud side should also be able to request specific actions to be carried out on the edge side. An example could be the starting or stopping of analytics running on the edge or the transfer to the cloud of a small snapshot of high-frequency data for further analysis. It is very important to apply white-listing and input validation to avoid exposing the edge to cyberattacks.

Edge computing normally subscribes to the Data Bus exposed by the edge gateway and consumes data for data processing. In some circumstances, edge computing invokes cloud or local servers through a web service to run advanced computational functions.

Edge computing must have the following features:

- The ability to perform an action on behalf of the IIoT middleware, either offline or online
- The ability to host custom applications

- The ability to run analytics in standalone mode, offline, in collaboration with the IIoT middleware, or in collaboration with the middleware of the local headquarters
- The ability to carry out actions or download analytics from the IIoT middleware
- The ability to send unstructured or a specific set of data to the IIoT middleware on demand or when triggered by a condition

The edge tools

Edge tools are a list of components that can be used to manage the edge easily; they include the command-line tool, remote human interface, logging, and antivirus. They should provide a user-friendly interface to analyze logs, manage patching, and configure adaptors. Edge tools work in collaboration with the Edge Manager to renew certificates and authorizations or to self-register during the initial setup.

The edge tools must have the following features:

- The ability to easily manage and configure data acquisition either remotely or locally

- The ability to register to receive patching and updates
- The ability to log which actions were carried out and by whom
- The ability to browse and trend the data using a UI
- The ability to self-configure and self-register to the cloud at startup
- The ability to receive and execute commands from the cloud

Looking at the functionalities that the edge must implement, we can understand that the IIoT edge is not a simple device, but requires sophisticated abilities in terms of performance, management, and computation. Next, let's understand the various edges that can be implemented.

Edge implementations

Cloud vendors or equipment manufacturer vendors are developing different solutions based on their own OSs or proposing cloud-agnostic software development kits (SDKs). We will take a closer look at some of these in the upcoming chapters, but in this section, we'll list the most important edges.

Azure IoT Edge

Azure IoT Edge is the edge solution proposed by Microsoft for Azure IoT. It supports store-and-forward, edge analytics, and several protocol adaptors to translate proprietary protocols or standard protocols to internet protocols. Azure IoT Edge also supports the OPC Server in both its implementations: OPC Classic and OPC UA.

AWS IOT Edge

Greengrass is the new generation of the IoT edge by AWS. AWS supports an SDK to build the AWS edge and extends cloud capabilities to edge devices with Greengrass. This enables them to carry out actions locally, while still using the cloud for management, analytics, and permanent storage. Greengrass supports OPC UA but not the OPC Classic interface. We will be discussing Greengrass in more detail in *Chapter 10* .

Node-RED

Node-RED is an agnostic framework that can be used to build a simple IoT edge visually. Node-RED also supports Raspberry Pi and other lite hardware. It is based on Node.js and supports several protocols and several standards. Node-RED also allows us to develop our own plugins. Node-RED is not specialized for industrial environments but can be used profitably for pilot projects and proof-of-concept projects, or in conjunction with other components.

In the following chapters (*Chapters 8* to *10*), we will understand three solutions with all the previously mentioned edges. Finally, we will cap off our exploration with hands-on laboratory experience.

Edge internet protocols

The edge has to communicate with the cloud or (in general) central system. The most important edge-to-internet protocols (see also *IoT and Edge Computing for Architects – Second Edition, Perry Lea, Packt Publishing*) are the following:

- MQTT and the secure channel MQTTS are the most important ISO protocols for machine-to-machine communication. The standard is ISO/IEC PRF 20922. We will explore MQTT in *Chapter 8* . MQTT is based on the publish/subscribe pattern.
- AMQP was developed for interoperability and messaging. AMQP is also based on the publish/subscribe pattern. It is similar to MQTT, but it is a heavier protocol.
- HTTP and HTTPS are not real protocols for IoT but are normally used in conjunction with the REST API to transmit data over the internet.
- The CoAP is a specific protocol for IoT, based on the REST API. For more information, please check out the following website: https://coap.space/ .

Now, we understand how to connect the edge to the cloud and what the most common protocols are. We will see practical examples with MQTT in *Chapters* 6 to *10* .

Implementing the IIoT data flow

In this section, we will discuss the IIoT data flow. We will look at it first from the perspective of its architecture and network, and then move on to considering the strengths and weaknesses of each available data source option. So far in this book, we have not yet seen a situation in which industrial equipment is directly connected to the cloud. This is actually technically possible—there are several pieces of industrial equipment that have this ability—using internet protocols, such as MQTT, CoAP, and AMQP. Examples of this kind of equipment include complex packaging systems, automated guided vehicles (AGVs), robotized working cells, or the devices and equipment involved in a fully automated warehouse.

In this section, however, we are interested in industrial plants in which the factory floor is built and managed by thousands of sensors and actuators. In such a scenario, it is truly not possible to connect all these signals directly to the cloud.

There are many reasons why this is the case:

- Variety : In industrial plants and factories, the technology used is always a mix of generations. We might find sensors that were installed just a year ago, using a real-time kernel, with self-diagnostic, auto-tuning, and signal-processing capabilities, together with very old sensors with limited or no fieldbus interfaces. It is not feasible to deal with these different technologies at the same time; we would have to implement different ad hoc connectivity solutions.
- Amount : The factory floor is made up of hundreds of sensors and actuators and the industrial process uses a huge number of microcontrollers and fully automated working cells. All these signals and information are needed to implement predictive and prognostic analytics on the cloud. They are also the basic building blocks to construct digital twins of the machines or processes. It's not possible to establish a direct connection to the cloud for each of these; the variety and amount of them would require too much maintenance and networking to be dealt with.
- Security : Last but not least, cybersecurity is a big barrier to implementing connectivity in a factory. Devices and equipment that are not designed to be resilient to cyberattacks will be much more exposed, thereby putting entire islands of automation at risk.

Let's now move on to looking at the industrial data sources and the related data-gathering techniques.

IIoT data sources and data

gathering

As discussed, PLC, DCS, SCADA, and Historian are the main data sources for the IIoT data flow. The following diagram shows the IIoT data flow schema, as mentioned in *Chapter 3* , *Industrial Data Flow and Devices* :

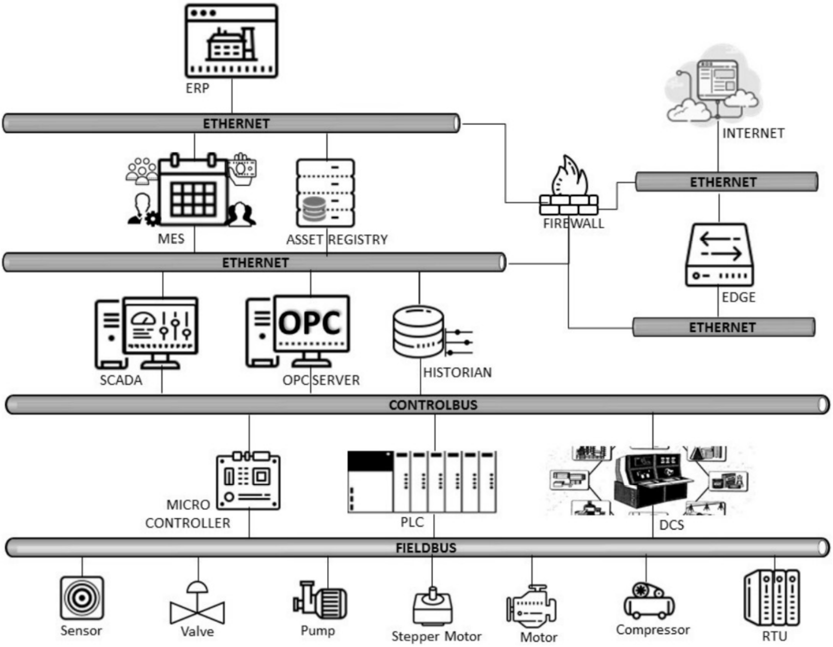

ERP
ETHERNET
INTERNET
MES
ASSET REGISTRY
FIREWALL
ETHERNET
ETHERNET
EDGE
ETHERNET
SCADA
OPC
OPC SERVER
HISTORIAN
CONTROLBUS
MICRO CONTROLLER
PLC
DCS
FIELDBUS
Sensor
Valve
Pump
Stepper Motor
Motor
Compressor
RTU

Figure 4.12 – IIoT devices and protocols in the factory

Let's start to analyze each of these in terms of their capabilities, their strengths and weaknesses, and their connectivity and data acquisition options.

PLC

PLCs receive data from sensors and send commands to actuators. In a large plant, it is likely that there will be several PLCs of different sizes with different capabilities. These are often structured hierarchically, with the highest acting as data concentrators. There aren't many PLCs that act as concentrators in a factory; you can often find them controlling an entire line of production or a specific functional area of the industrial plant. The PLC concentrators meet the needs of the IIoT data flow very well. They are very powerful and are usually linked to an OPC server gateway provided by the same vendor as the plant or the corporate network, thereby simplifying the task of transferring the data to the cloud over the internet. Sometimes, they are hosted on a separate board on the same rack as an

OPC UA server. Since OPC Classic is just run on the Windows box, the same option is not possible for the OPC DA server. If PLC concentrators are not available, we need to connect to the main PLC of every working cell, line production, or plant area from which we need to gather data. In a large plant, we hardly ever just use one PLC concentrator.

Advantages of the PLC

The PLC is a device that processes the signals coming from the sensors, the commands sent to the actuators, and the current status of the devices and equipment. Its fast scan time allows us to have fresh data coming from the sensors every 100 milliseconds or less. It is also reliable, steady, and deterministic. It has the fastest sampling rate and very few instances of downtime, which are planned well in advance. This is very important if we need to gather data without unplanned shutdowns or device reboots that might compromise the quality of the signals or the integrity of the data stream.

Disadvantages of the PLC

PLCs are at the first level of the automation pyramid. This means that using them to get data means working very close to the hardware, with a low level of abstraction. This brings the following problems:

- The PLCs, and the PLCs that act as data concentrators, manage, either directly or indirectly, several thousand input and output signals plus their internal data, calculations, and statuses generated by their control logic. In the context of the IIoT, all of these are tags, some of which come from physical instruments and others from internal calculation and intermediate logic steps. These often appear unstructured, or structured in a very basic way. They are rarely linked to a comprehensive data model, even though this should be one of the outcomes of a well-integrated IIoT platform. In any case, selecting the PLC as a data source means we have to deal with this complexity due to the low level of abstraction.
- Naming conventions are another obstacle to overcome since the names of tags might be different in different areas of the same plant. This might depend on the PLC vendor or the integrator that develops the control logic running on the PLC. Often, the naming conventions also reflect the naming of the physical instruments that are provided by the manufacturer of a specific plant area.

- We cannot simply connect a PLC to a device, such as the edge, that is exposed to the internet. This is not recommended by the integrated control system (ICS) security standards and, in any case, is not something that the field engineer would allow. We will be exploring how we can get around these security constraints further in *Chapter 5* , *Applying Cybersecurity* , but it is not always possible to do this for the following reasons:
    - The PLCs may be old and would therefore not be resilient to storm attacks on their Ethernet port.
    - There may be other old devices on the same local network that are potentially at risk.
    - There may be customer security policies to keep the control network as isolated as possible, due to the variety of the industrial devices that are linked to it and their vulnerability to cyberattacks.
    - The willingness of the team may also be a barrier; an automation engineer with a lot of experience in the field may be unwilling to accept the idea of connecting a PLC to the internet.
- Connecting directly to a PLC implies that we can talk to it according to its protocol. Although PLCs have become much more standardized due to the adoption of the fieldbus standard over the past few years, we would still need to develop, maintain, and keep updated the connectors for several protocols, including EtherNet/IP, PROFINET IO, DeviceNet, PROFIBUS DP, and MODBUS. This requires quite a lot of effort, especially if you consider that each of these protocols is liable to require enhancements and fixing, meaning you have to deal

with the compatibility and the backward compatibility of the developed connectors. Another option for using PLCs as a data source for the IIoT data flow is to connect to them through the OPC server. OPC UA, if available, is without a doubt the preferred choice compared to OPC Classic. OPC Classic, however, is preferred to a direct connection through the fieldbus protocol used by the PLC. If your choice is OPC Classic, try to run your OPC DA client on the same Windows box as the OPC DA server to avoid the DCOM issues discussed in the previous section.

DCS

From the perspective of the IIoT data flow, DCSs are very similar to PLCs, so all the previously analyzed strengths and weaknesses also apply in this case. There are just a few differences to underline, which are listed as follows:

- The architecture of the DCS is natively organized hierarchically. This means that in industrial plants where the automation systems mainly use DCSs, we don't find many of them and they are typically supplied by the same vendor. For example, in a medium-sized refinery, we might have three or four DCSs that manage the whole plant.
- The DCS plays the double role of controller and SCADA system. Many also have an integrated Historian natively connected to the DCS controllers. This helps to select the data to be gathered since they are usually organized and structured according to a data model. This

means we have to deal with less variety and complexity compared to using PLCs.

- The DCS, like the PLC, manages and collects thousands of sensors, actuators, internal calculations, and derived values and statuses. Typically, its scan rate is between 100 and 1,000 milliseconds. This is lower than the scan rate of a PLC but in most cases, the data is not connected by linking to its native protocol, but instead through its OPC server. Any OPC server is unlikely to provide data to its OPC clients with a refresh time under 1,000 milliseconds. This means that from the perspective of the IIoT data flow, it doesn't really matter whether the underlying device is a DCS or a PLC when gathering data from an OPC server.

SCADA

SCADA systems have the necessary drivers to communicate with the field devices of the factory floor through their legacy communication protocols or, more often, through OPC or a fieldbus protocol. Typical data acquisition intervals for SCADA applications are in the range of half a second to one minute.

A small-to-medium-sized SCADA system typically handles thousands of process signals, adopting a common approach for processing information based on a common set of parameters that are necessary for the correct management of the information. These parameters include the tag name, a description, the sampling time, a minimum and a maximum value, and engineering units.

Advantages of SCADA systems

One of the main functions of a SCADA system is the data acquisition system. This acts as a natural data concentrator, collecting data from different field devices, microcontrollers, and PLCs that control the different working lines or functional areas of the factory. From the perspective of IIoT data acquisition, SCADA systems are very similar to the DCS; they are organized hierarchically, there are few of them, and they are typically supplied by the same vendor. For example, in a medium-sized manufacturing plant, you might have one or two SCADA systems supervising the whole plant, plus some local HMI monitoring working in cells or small areas. Since SCADA adopts a common approach to processing industrial data that is often based on a common data model, it is easier to identify and recognize the data stream needed by the IIoT data stream. They may need to communicate with MES and ERP systems so they are usually already linked to the plant or corporate network. This simplifies the task of transferring data to the cloud over the internet.

Disadvantages of SCADA systems

SCADA systems are much less reliable than PLCs and DCSs. This is because of the following reasons:

- SCADA systems are updated quite frequently to add or change tags, units of measure, scripts, alarm thresholds, and synoptic. This means that SCADA applications have to be restarted often. They usually run on Windows OSs, so they are also frequently rebooted to apply security patches or other installation tasks.
- SCADA systems are built on several modules that execute different tasks. Sometimes, these might be overloaded and they may drop the communication tasks that are in charge of feeding the IIoT data

stream.

- Connecting directly to a SCADA system implies that we can talk to it according to its API or SDK. Although there are some SCADA systems, such as InTouch or iFIX, that are very common on the market, in any case, we have to develop, maintain, and update the connectors for the SCADA system. Just like for the PLC, this requires a considerable effort, considering that each SCADA system is liable to require enhancement and fixing, so you have to deal with the compatibility and the backward compatibility of the developed connectors.
- Again, similar to the PLC, another option for using SCADA systems as a data source for the IIoT data flow is to connect to it through its OPC server. The considerations about OPC interfacing that were mentioned in the section on the PLC are also valid here.

Historians

Historians have interfaces toward the most common industrial protocols and fieldbuses, including PROFIBUS, MODBUS, DeviceNet, and OPC, and sometimes also legacy protocols for old plants or DCS. These interfaces implement the most efficient ways to gather data according to the abilities of the specific protocol. To store large amounts of data with minimum disk usage and acceptable approximation errors, Historians rely on a data-filtering and compression engine. The values are stored according to the dynamics of the signal and their filtering and compression mechanisms. Signals coming from the same device, such as the pressure and the flow rate of a pump, could have, and typically do have, their row data stored in different timestamps. Typically, Historians store time-series data following a hierarchical data model.

Advantages of Historians

Historians, by definition, act as data concentrators. They collect data from different field devices, microcontrollers, and PLCs that control different working lines or functional areas of the plant, and SCADA systems. Historian databases are typically structured and linked to a structured asset model. This makes it easier to identify and recognize the data flow that the IIoT data stream is interested in.

Like SCADA, they need to communicate with the MES and ERP systems so they are usually connected to the plant or the corporate network, which again simplifies the effort of transferring the data to the cloud over the internet. In general, they are quite reliable compared to SCADA systems because their connectors, which gather data from different data sources, have built-in data buffering abilities and store-and-forward mechanisms. This means that, in general, they are quite resilient to network instability or unexpected downtime of the Historian server. The reliability of the data that is stored in the Historian depends almost entirely on the availability and reliability of the device from which the data comes.

Disadvantages of Historians

Historians were developed to collect data coming from the whole plant and often from all plants of the same company. In general, they are designed to manage and make available a large amount of data. This means that optimizations are made with respect to data sampling and storage. This might produce the following disadvantages:

- The data collected by the Historians might be a subset of the data collected by SCADA or the different PLC concentrators; sometimes, not all the data that is available with other data-gathering techniques is collected and made available by Historians.
- Data is stored using filtering and compression mechanisms. This implies that when we get data from Historians, it is stored as raw data, which does not provide a snapshot of an asset at a specific time as an outcome. This is something that we must deal with since typically, the data-stream analytics running in the cloud work with snapshots of data that are temporally consistent and related to an asset.
- The data-collection configuration of Historians usually implements a lower sampling rate than SCADA systems to minimize bandwidth consumption and optimize data storage. This means you might have a data collection time of a minute or even more. This is something that you have to take into consideration when developing analytics on the cloud.
- Connecting directly to a Historian system implies that we can talk with it according to its API or SDK. It has the same disadvantages for developing and maintaining the related connectors that we already analyzed for the PLC and SCADA. There are some Historians that are very common in the market, such as the PI System of the OSIsoft. Another option for using the Historian as a data source for the IIoT

data flow is to connect to it through its OPC server. In this case, if we choose to use OPC Classic, we must use the OPC Historical Data Access (HDA) server. This was born to exchange archived process data rather than real-time data, such as the OPC DA, so you can query for batches of aggregate data on a regular basis. The same considerations and suggestions that we explained in the previous section about how to deal with the OPC are also valid for the Historians.

In this section, we have summarized the most important industrial-side systems connecting machines, components, and assets. In the next section, we will learn about the deployment scenario of the edge.

Edge deployment and data flow scenarios

In this section, we are going to analyze five different options for connecting the edge to the industrial data sources mentioned previously, highlighting the strengths and weaknesses of each. The five options are as follows:

- The edge on a fieldbus

- The edge on OPC DCOM
- The edge on OPC-Proxy
- The edge on OPC UA
- OPC UA on the controller

These five options do not cover all possible scenarios of edge deployment in a plant network to gather data from a factory, but they are the main ones; all other possibilities can be derived from these or are combinations of them. In this section, we aren't going to take into account cybersecurity requirements or constraints, as the ICS standards and their associated best practices will be covered in *Chapter 5* , *Applying Cybersecurity* . Let's start by taking a look at the first option—the edge on fieldbus setup.

The edge on a fieldbus setup

In this setup, the edge is directly connected to the DCS or the PLC through a fieldbus protocol:

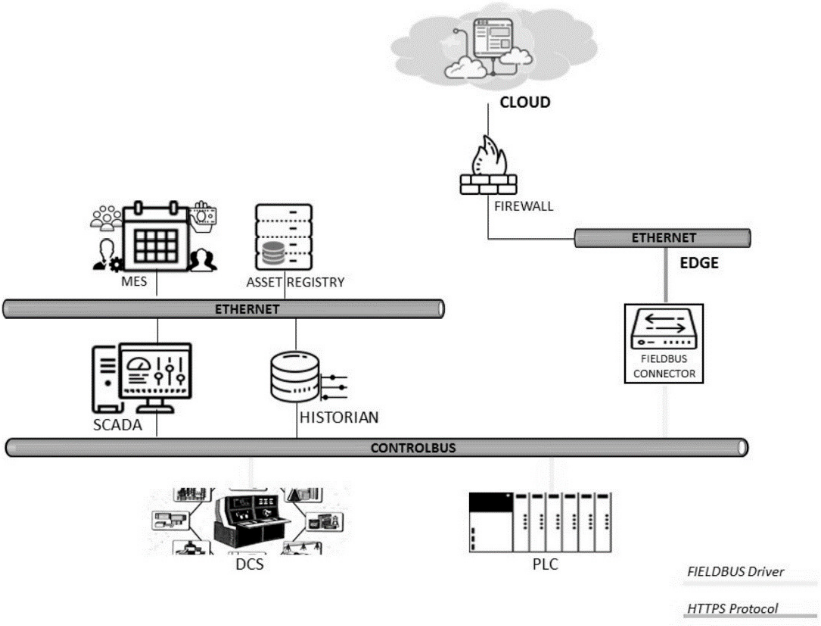

CLOUD
FIREWALL
ETHERNET
EDGE
MES
ASSET REGISTRY
ETHERNET
FIELDBUS CONNECTOR
SCADA
HISTORIAN
CONTROLBUS
DCS
PLC
FIELDBUS Driver
HTTPS Protocol

Figure 4.13 – Edge on fieldbus

This means that the edge device must implement and incorporate the connectors to the fieldbus protocols, such as EtherNet/IP, PROFINET, or MODBUS, in order to gather the data to be transferred to the cloud.

At first sight, this architecture seems very simple, but it's actually not that easy to implement and maintain. In fact, it has numerous strengths and weaknesses.

The *strengths* of this setup are the following:

- We have a direct connection to the PLC and the DCS, which provide the best data-acquisition mechanisms and the most accurate time-series databases, as we saw in the previous sections.
- One device implements both the data collection and the data transfer with no dependencies on other devices or applications. This means that there is just one point of failure and potentially better availability and reliability.
- There are just two interfaces with the external systems. The first is toward the controllers, the PLC, or the DCS, and is steady and reliable by design. The second is toward the cloud, which is also reliable due to the data-buffering capabilities and the store-and-forward mechanism implemented by the edge device.

The *weaknesses* of the fieldbus setup are as follows:

- There is a strong coupling with the controllers. The direct connection with the controllers forces the edge to implement and maintain a connector for each fieldbus protocol. This means it is susceptible to the previously discussed disadvantages.
- There is not a single PLC or DCS that controls the whole industrial plant. A PLC concentrator, like a DCS, controls just a single area of the plant. The related control networks are not linked with each other by design as they are deterministic networks. This would also make them vulnerable to security breaches. As we saw in the previous chapters, their information is joined and merged at a higher level by the SCADA and Historian systems. Because of this, an edge device must be deployed for each control network of the industrial plant from which

the data is to be gathered. The scattering of the control networks forces us to install more edge devices, leading to more complicated deployment and maintenance.

- In this setup, the edge device is connected at the same time to both the internet and the control network. According to the ICS standards, we should not connect a device that is connected to the internet directly to the control network; it should instead be placed in a demilitarized zone (DMZ). This means that the edge device should be separated by two firewalls, the first of which controls the interface toward the internet and the second of which controls the interface toward the control network. The isolation of the control network through firewalls is not easy. Special firewalls, called operational firewalls, are required, which have packet-inspection capabilities for the specific fieldbus protocol used. Even with these firewalls in place, it is common that customer policies and standards do not allow this type of connection.

The edge on OPC DCOM

In this scenario, the edge is not directly connected to the data source but to its OPC Classic server through an OPC Classic client:

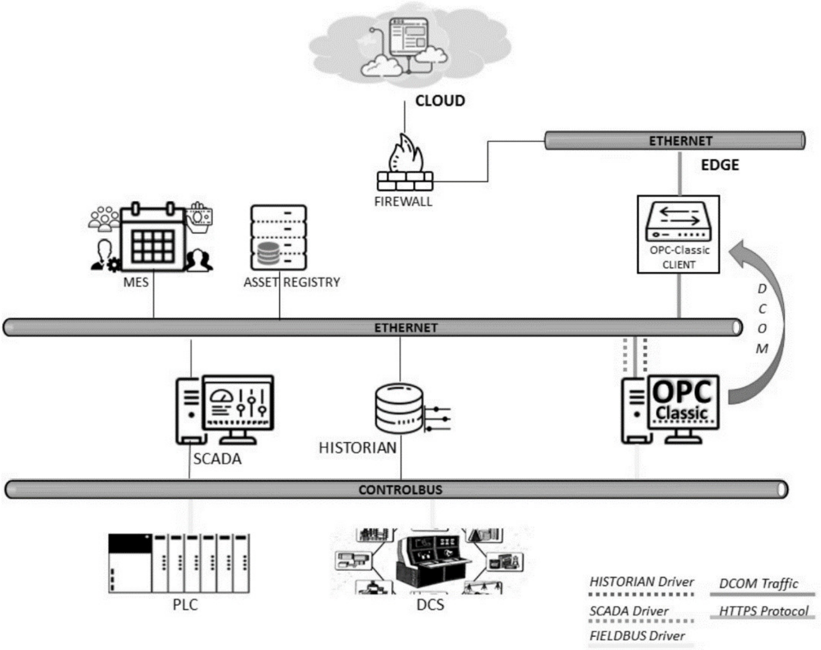

CLOUD
ETHERNET
EDGE
FIREWALL
OPC-Classic CLIENT
MES
ASSET REGISTRY
ETHERNET
D C O M
OPC Classic
SCADA
HISTORIAN
CONTROLBUS
PLC
DCS
HISTORIAN Driver
DCOM Traffic
SCADA Driver
HTTPS Protocol
FIELDBUS Driver

Figure 4.14 – The edge on OPC DCOM

This means that the edge must implement and incorporate the OPC Classic client interface. The data source could be one of those mentioned previously—a PLC, a DCS, a SCADA system, or a Historian. The OPC abstracts the data source, which is why, in the preceding diagram, the OPC server has a dotted link with each of the potential data sources.

Obviously, the OPC server must be specific to a particular data source, but, from the perspective of the client and the edge, this does not matter. Whichever data source is connected, the OPC server will expose its data through an OPC Classic interface to which the edge can connect through its OPC Classic client interface. This is the OPC DA client interface for the PLC, DCS, and SCADA systems, and the OPC HDA interface for the Historian system. According to the underlying data source abstracted by the OPC server, we will have the related advantages and disadvantages that were discussed and analyzed in the previous section.

Strengths of the edge in OPC DCOM

The strengths of the edge in OPC DCOM setup include the following:

- The OPC interface abstracts the underlying device that acts as the data source. This means that there is no need to implement and maintain a connector for each fieldbus protocol to which the edge connects.
- The OPC server is typically linked to more than one PLC or DCS that has the same underlying industrial protocols. This means that a single OPC server can collect data coming from more than one PLC, thereby

simplifying the edge deployment.

- Since the edge is not connected directly to the control network, it is easier to deploy it in a DMZ, as stipulated by the ICS standards.

Weaknesses of the edge in OPC DCOM

The weaknesses of this setup include the following:

- There is DCOM communication between the edge and the OPC server, which means we have to deal with DCOM issues related to configuration, performance, and maintenance.
- Since the edge uses the OPC Classic interface, it can only run on a Windows box.
- For security reasons, the edge and the OPC server must run on different networks that are segregated by means of firewalls. By default, however, DCOM assigns ports dynamically from the TCP port range of `1024` to `65535`, and it is not easy to restrict the number of the TCP ports used by DCOM to allow it to cross traffic through the firewalls.

The edge on OPC-Proxy

In this setup, the edge is not directly connected to the OPC Classic server but is instead connected through an OPC-Proxy by means of its OPC UA client interface:

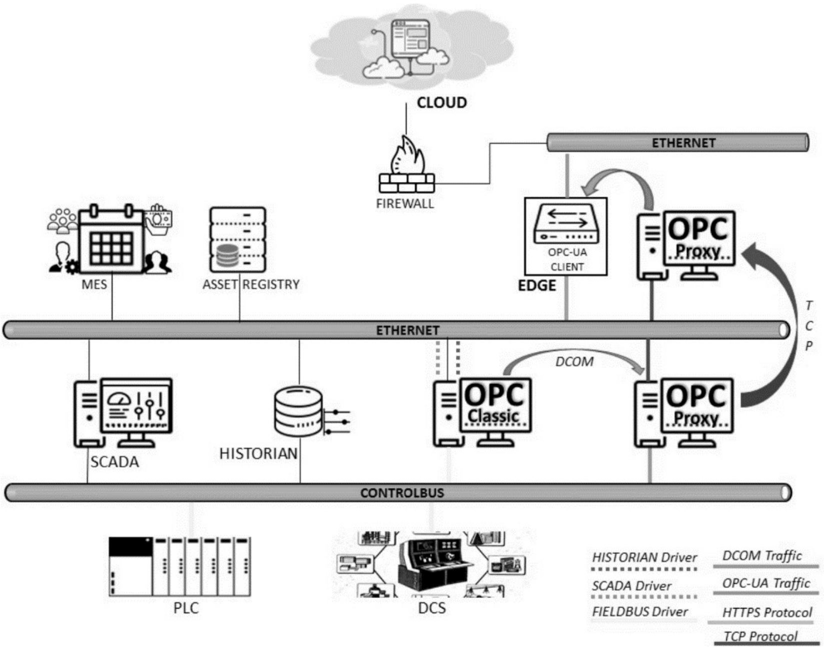

CLOUD
FIREWALL
ETHERNET
OPC-UA CLIENT
EDGE
OPC Proxy
MES
ASSET REGISTRY
ETHERNET
T C P
DCOM
SCADA
HISTORIAN
OPC Classic
OPC Proxy
CONTROLBUS
PLC
DCS
HISTORIAN Driver
SCADA Driver
FIELDBUS Driver
DCOM Traffic
OPC-UA Traffic
HTTPS Protocol
TCP Protocol

Figure 4.15 – OPC-Proxy scenario

An OPC-Proxy enables the OPC Classic server, which is based on the COM/DCOM architecture, to be connected with the OPC UA products and can be used to manage the traffic in both directions at the same time. It typically wraps an OPC UA server that gets data from its internal COM client. The wrapper is needed to connect the OPC UA client to the OPC Classic servers. As a proxy, it supports the traffic in the opposite direction. The proxy is a COM-based server that gets its data from its internal UA client.

The proxy is needed to connect OPC Classic clients to OPC UA servers. It can also be used for firewall tunneling a COM/DCOM-based OPC through a secure OPC UA connection.

In this setup, the OPC Classic server collects data from the related underlying device. The data gathered through DCOM from the OPC-Proxy linked to the same network of the OPC server is tunneled in a TCP channel across the network boundary to reach the other OPC-Proxy on the other side of the TCP channel. This OPC-Proxy exposes the data through its OPC UA internal server, therefore making it available to the OPC UA client interface, which implements and incorporates it into the edge to be later transferred to the cloud. Like the previous scenario, in this setup, the data source could be a PLC, a DCS, a SCADA system, or a Historian, since the OPC abstracts the data source.

Strengths of the edge on OPC-Proxy

The strengths are the same as those of the previous one, plus the following:

- There is no need to move the DCOM traffic between firewalls since this occurs solely between the OPC server and the OPC-Proxy in the same network. Thanks to the tunneling, just the TCP traffic between the two OPC proxies on opposite sides must be managed by the firewalls.
- The OPC-Proxy abstracts the OPC interfaces. The edge that is implementing the OPC UA client interface can perform OPC UA reading requests even if there is an OPC Classic server on the other side, and vice versa.
- The OPC-Proxy can act as an OPC concentrator, gathering the data from several OPC Classic or UA servers and making it available through a unique endpoint.
- The edge only has to implement an OPC UA client interface, so it doesn't have to run on a Windows box.

Weaknesses of the edge on OPC-Proxy

The weaknesses of this setup include the following:

- The OPC-Proxy implements the OPC Classic interface so it can only run on Windows boxes
- The OPC-Proxy extends and makes the acquisition chain more complex by introducing additional hosts and applications, therefore increasing the potential points of failure and the related maintenance necessary

The edge on OPC UA

A direct connection to an OPC UA server is definitely the preferred scenario as this allows us to make the most of the features and capabilities of OPC UA. It can be deployed in two different ways. In the first setup, the edge is directly connected to the OPC UA server by means of its OPC UA client interface:

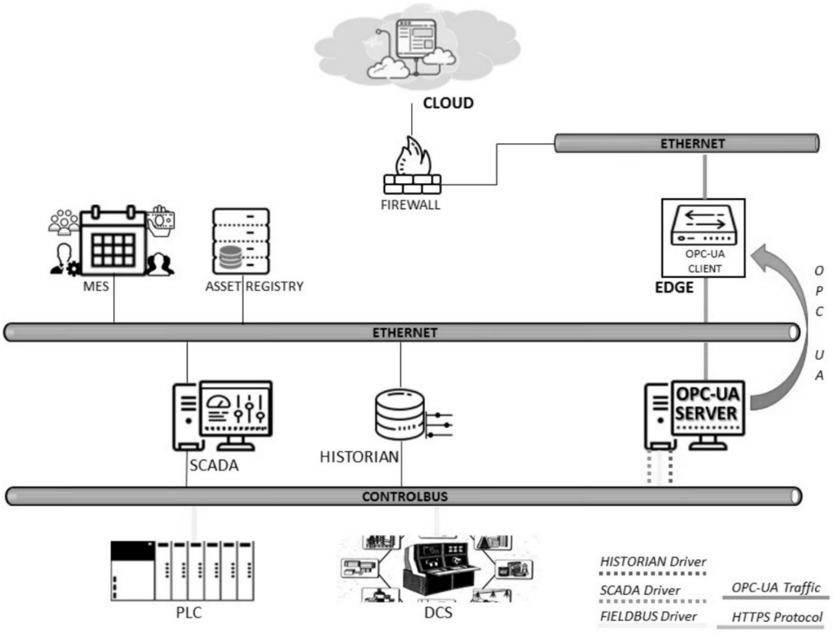

CLOUD
ETHERNET
FIREWALL
OPC-UA CLIENT
EDGE
MES
ASSET REGISTRY
ETHERNET
OPC UA
OPC-UA SERVER
SCADA
HISTORIAN
CONTROLBUS
PLC
DCS
HISTORIAN Driver
SCADA Driver
FIELDBUS Driver
OPC-UA Traffic
HTTPS Protocol

Figure 4.16 – The edge on OPC UA

The data source can be one of the ones analyzed previously: a PLC, a DCS, a SCADA system, or a Historian. OPC abstracts the data source, which is why, in the preceding diagram, the OPC UA server shows a dotted link for each of these potential data sources. Obviously, in this scenario, like in the previous one, the OPC server must be specific for that data source, but from the client's perspective and the point of view of the edge, this does not matter.

Strengths of the edge on the OPC UA

The strengths of the edge on the OPC UA scenario include the following:

- The OPC interface abstracts the underlying device that acts as a data source. This means there is no need to implement and maintain a connector for each fieldbus protocol that the edge connects to.
- The OPC UA server is typically linked to more than one PLC or DCS that has the same underlying industrial protocols. A single OPC server can collect the data coming from more than one PLC, therefore simplifying the deployment of the edge. Moreover, the OPC UA server can act, through its Discover Service function, as an OPC UA server

concentrator. This means that it can also expose the data coming from OPC UA servers that are linked to different underlying industrial protocols.

- Since the edge is not connected directly to the control network, it is easier to deploy it in a DMZ, as stipulated by the ICS standards.
- There is no COM or DCOM communication between the edge and the OPC UA server. This means that there is no need to deal with the DCOM communication mechanism based on TCP dynamic ports.
- Since the edge must implement just the OPC UA client interface, it is not restricted to running just on a Windows box.

Weaknesses of the edge on OPC UA

The weaknesses of this setup include the following:

- The OPC UA server will gather data from different data sources. For this reason, it becomes a crucial device from the perspective of the IIoT data flow. If it experiences downtime, this implies the entire IIoT data flow breaks and we lose data from all data sources to which the OPC UA server is connected.
- In case of partial data loss, it is not easy to understand and identify which underlying data sources are involved. It is also difficult to identify the issue causing the interruption.

OPC UA on the controller

In the second setup related to OPC UA, the edge is connected to the OPC UA server for the fieldbus protocol used by the PLC or DCS vendor. It runs on a specific board that can be installed in the rack of the controller itself. In OPC UA on controller setup, the edge is connected to the OPC UA server that is hosted by the controller by means of its OPC UA client interface:

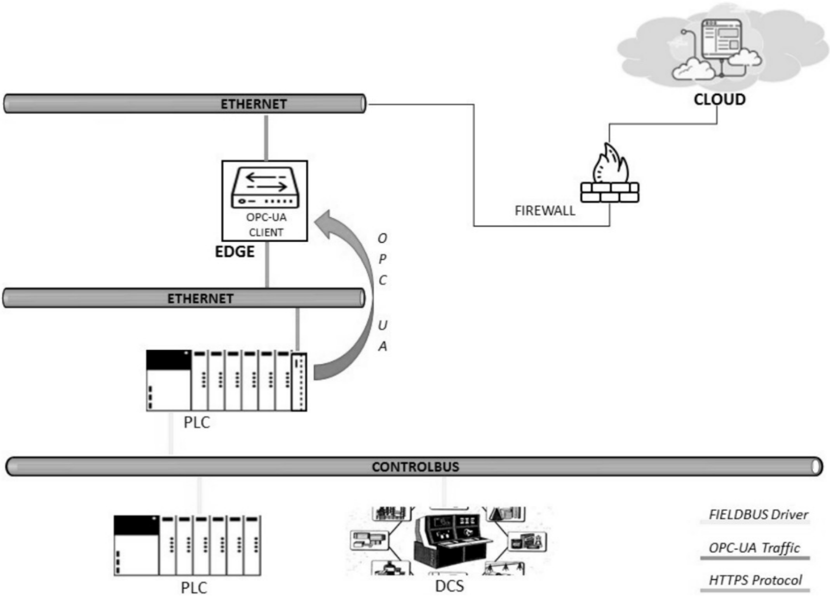

CLOUD
ETHERNET
OPC-UA CLIENT
EDGE
FIREWALL
O P C U A
ETHERNET
PLC
CONTROLBUS
PLC
DCS
FIELDBUS Driver
OPC-UA Traffic
HTTPS Protocol

Figure 4.17 – OPC UA server controller scenario

The hosting of an OPC UA server directly by the controller is an option that several PLC vendors are starting to offer. Typically, the OPC UA board has a

Linux kernel that hosts the OPC UA server, which is connected to the controller by means of its internal bus. This setup, like the previous one, uses the OPC UA interface. In terms of its strengths and weaknesses, therefore, it is very similar. There are a couple of main differences, which are listed as follows:

- The OPC UA server running on the controller only makes the data of that controller available. Due to the segregation and isolation of the control networks, an edge device would not be able to reach all the OPC UA servers that are hosted by the different controllers.
- There are no issues involved with deploying a firewall between the edge and the OPC UA server running on the controller. The OPC UA traffic is made up of TCP or HTTPS traffic, which is quite easy to manage by means of the firewall policies. On the other hand, the OPC UA server doesn't use authentication or allow its OPC UA clients to open a communication channel if its security mode is set to **`None`**. As a consequence, it might not be easy to convince field engineers to connect the PLC to a device, such as the edge connected to the internet.

We now understand how to connect the edge, and therefore the cloud, via OPC to the industrial environment. We will see practical examples in

Chapters 7 to *10* .

Summary

In this chapter, we learned how to implement the IIoT data flow in a complex industrial plant. We started by looking at OPC, including what it is, how it has evolved, and how it works in both its implementations—OPC Classic and OPC UA. We explored how it connects to data sources to expose their data through common interfaces, following a common data model. We then went into detail about the edge, analyzing its features, internal components, architecture, interfaces, and current implementation.

After that, we started our journey on the IIoT data flow. We looked at how to select a data source to connect to in order to gather data. We explored controllers, such as the PLC or the DCS, a SCADA system, and a Historian system. Each of these options has its own advantages and disadvantages; no one is better than the others. The choice must be made depending on the capabilities and constraints that exist in our plant and the requirements of our specific use case.

We then proceeded on our journey through the IIoT data flow by analyzing the different setups for connecting the edge to selected industrial data sources. We looked in depth at five different setups, highlighting the strengths and weaknesses of each. In general, setups that use the capabilities of OPC UA are preferable, but this is not always possible, especially when dealing with old equipment. In cases such as this, the OPC Classic, with or without OPC-Proxy, is the only option.

In the next chapter, we will learn how to secure these edge deployment setups from the perspective of cybersecurity.

5

Applying Cybersecurity

Cybersecurity is a very hot topic and is becoming increasingly important, not just from an economic perspective but also politically and socially. The economic impact of cybersecurity is easy to understand – if information technology infrastructure is compromised in some way, it is logical to expect some kind of economic impact. What is not so apparent is the importance that cybersecurity has from a political and social perspective. The Cambridge Analytica scandal is representative of the importance of cybersecurity from a political perspective. In this case, the personal data of several million Facebook users was used, or at least attempted to be used, to influence the US elections. If you take a look at international airports, the fights that they are battling have more to do with cyber attacks than traditional issues. Many companies manage and own tons of personal data. Analysis of this issue would require a book in itself and is beyond the scope of this book, but it is important to bear this in mind when discussing cybersecurity and data protection.

In an industrial environment, cybersecurity relates to business continuity and intellectual property. This chapter will discuss the IIoT data flow from a cybersecurity perspective, restricting the analysis to the industrial control system and the IIoT data flow described in the previous chapter. We will discuss what it means to have a cyber strategy and how this applies in the industrial environment. We will also outline the Defense in Depth (DiD) approach. In order to better understand the architecture to segregate a control network and secure the IIoT data flow, we will also provide some relevant information about firewalls.

In this chapter, we will cover the following topics:

- What is the DiD strategy?
- DiD
- DMZ, firewalls, and their classes
- Common architectures to secure control networks
- SIEM and ICS cyber forensics
- How to secure the IIoT data flow

What is a DiD strategy?

Today, all companies have been effectively forced to consider the risks they face and evaluate the vulnerability of their assets, as regards their related potential economic impact. Once risks have been assessed, priorities can be established and a defense strategy can be arranged. This means that we need to adopt DiD logic, without being discouraged by the apparent difficulty of this approach. But what is DiD?

DiD is an approach to information security by which a security posture is achieved through the coordinated and combined use of multiple security countermeasures. It is based on the integration of three different categories of elements – people, technology, and operating methods. The redundancy and distribution of countermeasures are based on two main concepts – defense in multiple places and layered defenses.

These concepts are certainly not new; they come from the military. We assume that if an attack is successful, this represents a failure of a security mechanism, and other security mechanisms should intervene to allow for the adequate protection of a whole system. This is quite logical if we understand that, for an enemy, it is more complicated to penetrate a complex defensive structure that is made up of several levels than a single barrier. This concept is sufficient to provide each organization or company with technologies, solutions, and technological, operational, and organizational measures and countermeasures. However, this also leads to an exponential increase in complexity and costs, without any guarantee of the actual results and even the risk of counterproductive effects. How the security of company information should be dealt with must be established by evaluating the trade-off between risk and expected profit.

Let's now examine, one by one, the three key elements that comprise DiD – people, technology, and operating methods.

Making people aware

In a company, it is a good idea for information security to be the responsibility of people at a high level. Senior management must have an in-depth knowledge of cyber risks and threats. This awareness is the foundation for the following actions:

- Issuing policies
- Defining procedures
- Implementing physical security measures to protect technological infrastructure
- Assigning roles and responsibilities
- Training personnel

Understanding technology

DiD involves the spread of technological barriers, a multi-level approach, and the implementation of mechanisms that detect intrusions.

The most common technological components are the following:

- Firewalls
- Antivirus and/or anti-spyware software
- An Intrusion Detection System (IDS) and Intrusion Prevention System (IPS); these can either be integrated into firewalls or added as independent systems
- Cryptography
- The definition and application of clear rules to access systems
- A hierarchical password system and software that monitors the

integrity of the files

Authentication can occur using techniques linked to biometrics. In this context, Public Key Infrastructure (PKI) can be implemented at the company level to manage the access keys to the various components of the infrastructure robustly.

> ### *Protection from intrusions*

> *It is important to emphasize the aspects related to detection that support protection from intrusions. These must provide, besides the analysis of activities, the answer to these questions: Am I under attack? What is the source of the attack? What is the goal of the attack? Which countermeasures can I put in place?*

Operating modes and procedures

The daily activities of the members of a company must adhere to the same objectives as DiD. In this context, the following are among the most important elements:

- The application of security updates and the continuous updating of antivirus solutions

- The maintenance of an Access Control List (ACL), which establishes the rules to access the identified critical systems
- The execution of assessment activities, such as periodic scans on system vulnerabilities, monitoring, and the consequent reaction to threats
- The planning of disaster recovery and business continuity

The preceding list of activities is not exhaustive. There are many tools that can be used, including reactive and proactive techniques, preventive measures and remediation techniques, forensics, and even intelligence techniques.

Given the breadth of the solutions, following the directions of DiD in an indiscriminate style could increase the complexity of a whole system, violating the principle of simplicity, which is very often touted as a best practice in a security environment. The addition of new security and security features increases complexity, which, paradoxically, entails new risks. What must guide the decision-making process in a business environment is, as always, a balance, but how do we go about making the correct choices? The answer lies in risk assessment. The priorities for investments in the security sector must be dictated according to the risks to a company.

The DiD in an industrial control system environment

Even if the concepts behind DiD are not new, they have only been applied to Industrial Control System (ICS) environments recently. Historically, the majority of organizations held the belief that legacy ICSs were impervious to cyber attacks due to their isolation from Information Technology (IT) networks and the implementation of physical security measures. However, the merging of IT and ICS architectures has challenged this notion, and the ever-increasing need to use data and information produced on the factory floor has highlighted the potential risk that control systems are now exposed to.

DiD encompasses more than a single entity; it is a fusion of individuals, technology, procedures, and, crucially, an awareness of potential adversaries. Even the most advanced technology cannot eliminate human error. Applying DiD strategies to ICS environments improves security by making intrusions more difficult and, simultaneously, improving the probability of detection and the ability to defend against a malicious threat.

To minimize an adversary's chances of success and limit their attack surfaces, DiD security employs various strategies and solutions. Some effective options are outlined in the following list. Refer to the *Further reading* section of this chapter for a more in-depth explanation of all of these:

- A risk management program to identify and characterize risks.
- Maintaining the asset inventory.
- Cybersecurity architecture :
    - Defining a set of standards/recommendations, policies, and procedures to define architectures

    - Defining a list of assets to grant physical security; the following is a list of the common use cases
    - Field electronics with a key access locked down
    - Access controls and physical security ICS network architecture

- Common architectural zones:
    - Demilitarized zones (DMZs)
    - Virtual LANs

- ICS network perimeter security : The security measures put in place to protect network infrastructure that supports industrial control systems from unauthorized access, malware, and other cyber threats. This mechanism leverages firewalls and remote access and authentication.
- Host security defines the following :
    - How to handle patch and vulnerability management
    - How to maintain and define access policies for field devices
    - How to maintain and define access policies for hypervisor systems

- Security monitoring includes the following :
    - Intrusion-detection systems
    - Security audit logging

    - Security incident and event monitoring
    - Vendor management
    - Supply chain management
    - Managed services/outsourcing
    - Using cloud services

- The human element : Defining policies, procedures, training, and best practices for people (e.g., administrators, operators, and developers). It is also very important to define metrics at the company level to monitor the level of awareness of cybersecurity concepts.

Firewalls

One of the best practices of a DiD strategy is to isolate the Control Network (CN), which is also often called the Process Control Network (PCN), from the corporate and internet systems using firewalls. In the realm of industrial CNs and PCNs, the implementation of firewalls is a relatively new development. Most IT firewalls are not designed to recognize industrial-control protocols and can introduce undesirable latency into time-sensitive systems. Firewalls, being intricate pieces of technology, can pose operational challenges (for instance, maintenance of a highly distributed secure network is very tiring) that differ from those encountered in the IT sector. Their optimal performance and security reliance necessitate meticulous planning, configuration, and maintenance. In this section, we are going to take a look at some basic information about firewalls, as well as how they are usually deployed in a factory to segregate a CN and protect industrial devices.

Basically, a firewall is a component (hardware or software) used to filter, control, and monitor traffic to and from a network, protecting the devices of that network. It checks for the traffic passing through it and ensures that the network messages fit predefined security criteria or policies. Messages that don't meet the policies are discarded. It can also be considered a filter that blocks unwanted network traffic and forces specific constraints and/or restrictions on the amount and type of communication that occurs between a secured network and other networks. Firewalls can exist in different shapes and configurations. A network firewall can take various forms, including a physical device connected to a network, a virtual appliance with firewall capabilities on a hypervisor, or a software solution installed on a host. These distinct hardware and software options are commonly known as *network firewalls* and offer the most comprehensive security and management capabilities. From now on, when we use the term *firewalls* , we are referring to this latter kind.

Network traffic is sent in discrete sequences of bits, called packets. Each packet contains separate pieces of information, including, but not limited to, the following:

- The IP address source and destination
- The port number of the source service and the destination service
- The protocols that are accepted or denied

- The direction of the traffic – inbound or outbound

A firewall, upon receiving a packet, analyzes the preceding characteristics to establish the proper action to take. It may drop the packet, buffer it temporarily to limit the bandwidth usage according to a class of service, or forward it to a different recipient. The firewall is based on a set of rules. These rules are commonly referred to as ACLs. There are five different classes of firewalls, depending on their analysis and action capabilities; in the following sections, we will explore the main ones.

A packet filtering firewall

The configuration of this firewall is based on a set of basic policies (block/allow) and a series of specific rules for the involved source and destination IP addresses and ports. A packet filtering firewall analyzes the headers of each IP packet in transit for the following information:

- The source and destination IP addresses
- The encapsulated transport protocol (for instance, TCP or UDP)
- The source and destination ports of the transport protocol
- The inbound and outbound traffic

A stateful inspection firewall

This works like a packet filtering firewall, but it also tracks all TCP connections that are open from and to the outside. It explicitly models itself on the TCP session concept, allowing us to define rules on this basis. For example, we can automatically accept all packets that come from a previously authorized TCP session.

Application proxy firewall

In the application proxy firewall, there is no direct connection between the inside and the outside of network machines; instead, there are two separate connections. The proxy works at the application level, just like the Hypertext Transfer Protocol (

HTTP) or the File Transfer Protocol (FTP). It receives requests according to these protocols and forwards or blocks them, depending on the configuration. This means that all internal machines and clients are forced to go through the proxy, since direct access to the external servers is blocked.

Deep Packet Inspection (DPI) firewalls

These are an emerging trend in the firewall domain. They typically offer filtering deeper into the application layer than a traditional application proxy, by allowing or blocking packets according to their semantics. They specialize in understanding industrial protocols. For example, a DPI firewall for the OPC Classic protocol could be configured to only allow reading of an underlying device, blocking any attempt at writing.

Goals of a firewall

Basically, the goal of firewalls is to mitigate (or annihilate) the risk of unauthorized access and unwanted network traffic to internal devices on a PCN. The risk minimization strategy is based on a few golden rules:

- There should be no direct connection from the internet to any device linked to a PCN
- Access from the corporate or plant network to the PCN must be restricted to what is really needed after an in-depth analysis of the possible alternatives
- The remote support of control systems should only be allowed if secure methods of authorization are in place
- If wireless devices are used, secure connectivity must be implemented
- Rules and policies must be well-defined, indicating the type of traffic allowed between networks
- There should be regular monitoring of the traffic coming in and going out of the PCN
- There should be a secure communication channel to manage the firewall

In this section, we learned about the basic components for securing an IT and OT network. In the next sections, we will study some of the most common IIoT architectures.

Common control-network-segregation architectures

This section outlines the most common security practices that are currently used in industrial-control environments in terms of the architecture, design, deployment, and management of firewalls to separate a PCN from a corporate network.

The solutions presented are related to two main scenarios:

- Designs that utilize two-zone firewall-based designs without a DMZ
- Designs that utilize three-zone firewall-based designs with a DMZ

> ### *Dual-homing*

> *There is also another scenario, which is often referred to as dual-homing. In this scenario, dual-network interface cards are installed either in a workstation or a control device that requires access to both the corporate network and PCN. Dual-homing is an easy way to connect the corporate network to the PCN, but it puts the dual-homed device at a significant security risk, even if a personal firewall is installed on it. For this reason, it is rarely used and will not be examined in this section.*

Network separation with a single firewall

The simplest architecture for network separation involves a single firewall. In the scenario shown in the following diagram, a simple two-port firewall is introduced between the corporate network and PCN, with a significant security improvement:

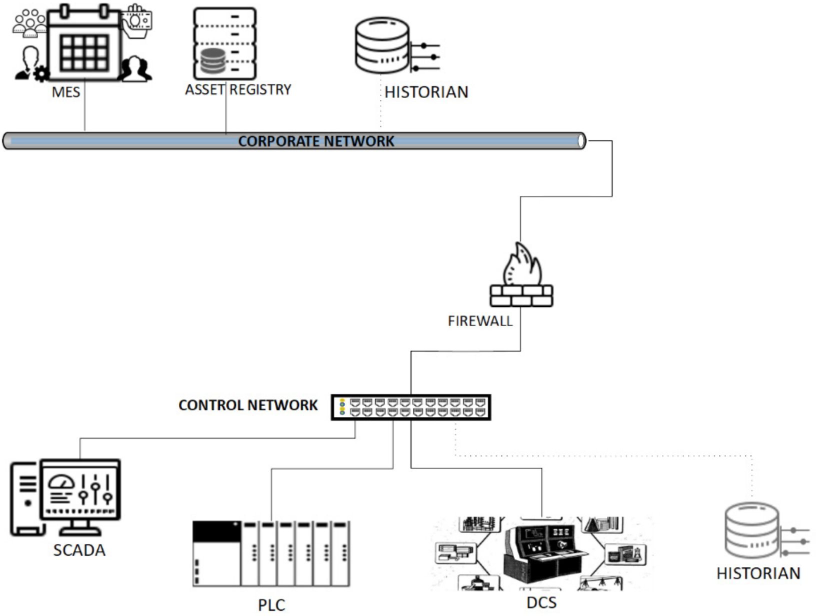

MES
ASSET REGISTRY
HISTORIAN
CORPORATE NETWORK
FIREWALL
CONTROL NETWORK
SCADA
PLC
DCS
HISTORIAN

Figure 5.1 – Network separation with a single firewall

Firewalls available on the market provide stateful inspection for all TCP and UDP packets, as well as application proxy services for popular internet application protocols, including FTP, HTTP, FTP, and SMTP. If they are well configured, they can significantly mitigate the risk of a successful external attack on a CN.

Even so, in this scenario, there is an issue related to which network contains the servers that need to be shared between the corporate network and CN. For this reason, the data historian in the preceding diagram appears on both networks but is grayed out on the CN. If a shared server, such as the data historian or an OPC server, resides on the corporate network, a rule must exist within the firewall that allows the historian or OPC traffic to communicate with the control devices on the CN. A packet originating from a malicious or incorrectly configured host on the corporate network and appearing as a legitimate packet (e.g., data historian or OPC) would be forwarded to individual PLCs.

If the shared server resides on the PCN, a firewall rule must exist to allow all hosts from the corporation to communicate with it, putting the shared servers at risk of exploits or spoofing.

A firewall with a DMZ

This is often useful to establish a zone that doesn't belong to anyone and exists only as a haven for data or applications. Such a space is often called a DMZ, borrowing again from the lexicon of the military. To securely implement a DMZ, firewalls with a minimum of three interfaces are necessary, as opposed to the standard configuration of public and private interfaces. One interface is linked to the corporate network, another to the perimeter network (PN), and the remaining interfaces are used to connect to the devices or resources intended for sharing. This scenario is shown in the following diagram:

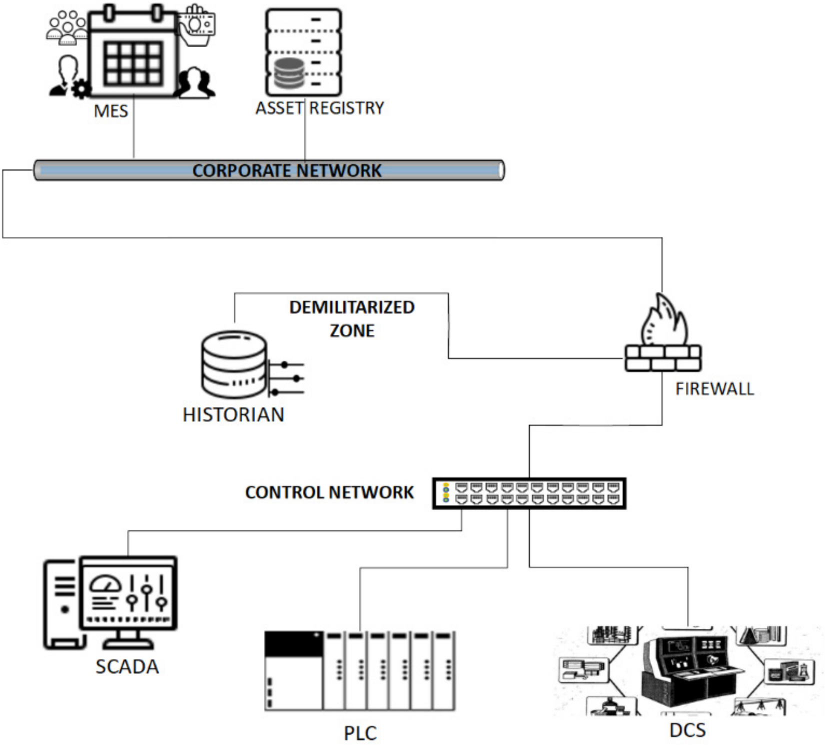

MES
ASSET REGISTRY
CORPORATE NETWORK
DEMILITARIZED ZONE
HISTORIAN
FIREWALL
CONTROL NETWORK
SCADA
PLC
DCS

Figure 5.2 – A firewall with DMZ

By placing shared devices or resources in the DMZ, no direct communication is required from the corporate network to the CN. Most firewalls allow for multiple DMZs to forward traffic between zones according to specific rules. The firewall does not allow arbitrary packets from the corporate network to pass to the CN. The traffic from other network zones is regulated by it as well.

A paired firewall with a DMZ

An alternative architecture to the firewall-with-DMZ solution is to use a pair of firewalls to build up the DMZ, linked to each other and positioned between the corporate network and PCN. The resources and devices to be shared are deployed in the DMZ between the firewalls, as shown in the following diagram:

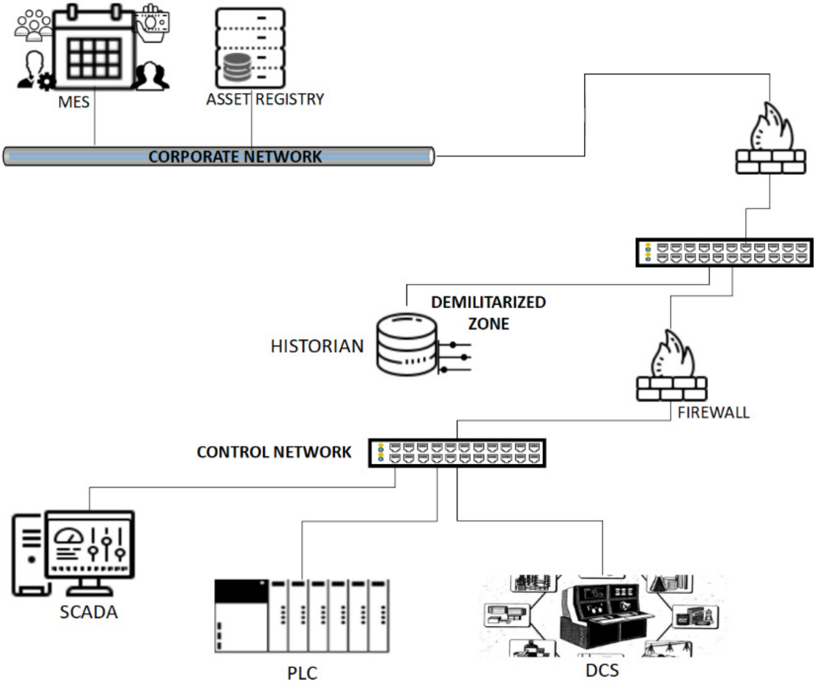

MES
ASSET REGISTRY
CORPORATE NETWORK
DEMILITARIZED ZONE
HISTORIAN
FIREWALL
CONTROL NETWORK
SCADA
PLC
DCS

Figure 5.3 – A paired firewall with DMZ

In this scenario, the firewall positioned in the corporate network blocks the arbitrary packets from proceeding to the CN or the shared historians. The other firewall prevents unwanted traffic from a compromised server from entering the CN.

A firewall with DMZ and VLAN

In a LAN, all the hosts belonging to the same physical infrastructure (such as switches, hubs, or cables) are part of the same IP network. If multiple IP networks are needed, several physical infrastructures are required. A Virtual LAN (VLAN) allows us to overcome this constraint, and different logical networks can share the same physical infrastructure. The VLAN is an independent IP subnet that can share the same physical network as other VLANs.

In certain instances, specific functional regions or operational units within a control system do not require inter-area communication and can even benefit from its absence. By dividing and isolating different sections into various VLANs, inter-VLAN communication becomes feasible. This communication can be regulated through the application of basic packet filters, using both layer 3 and layer 2 switches, as depicted in the subsequent diagram:

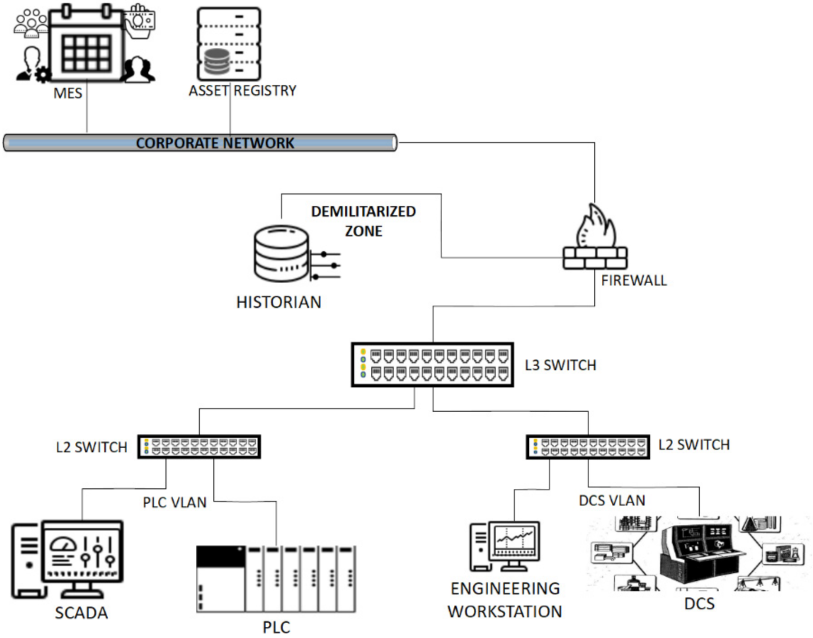

MES
ASSET REGISTRY
CORPORATE NETWORK
DEMILITARIZED ZONE
HISTORIAN
FIREWALL
L3 SWITCH
L2 SWITCH
L2 SWITCH
PLC VLAN
DCS VLAN
SCADA
PLC
ENGINEERING WORKSTATION
DCS

Figure 5.4 – A firewall with DMZ and VLAN

In this scenario, VLANs help to prevent the propagation of unwanted traffic across the whole CN. In this section, we learned about some architectures; in the next sections, we will study how to make an IIoT data flow secure.

Securing the IIoT data flow

In *Chapter 4* , we analyzed five different options to connect the edge to industrial data sources, highlighting the strengths and weaknesses of each. The five options that we analyzed were the following:

- The edge on a fieldbus
- The edge on OPC Distributed Component Object Model (DCOM)
- The edge on OPC Proxy
- The edge on OPC UA
- OPC UA on a controller

We have not yet considered the cybersecurity requirements and constraints for each of these options. In this section, we will understand how to secure them from a networking perspective, according to the standards of the ICS and the related best practices. As we outlined in the previous *Common control-network-segregation architectures* section, securing a CN is just one of the recommendations of the DiD strategy that can be used to mitigate the cyber risks of entire control system environments. There are other best practices and specific countermeasures to implement to create an aggregated, risk-based security posture, defending control systems against cybersecurity threats and vulnerabilities. Such analysis would require its own book and is beyond our scope here. For this reason, we have restricted our analysis of mitigating cyber risks in the ICS to the network architecture, since this plays a key role in the IIoT data flow.

For each of the preceding five options, the starting point will be the network schema that we previously discussed for the related edge deployment. It will be modified to segregate the CN, mainly through the creation of DMZs that host the shared devices.

Let's start by looking at the edge on a fieldbus setup.

Securing the edge on a fieldbus

In this scenario, the edge is directly connected to the DCS or PLC through a fieldbus protocol, as shown in the following diagram:

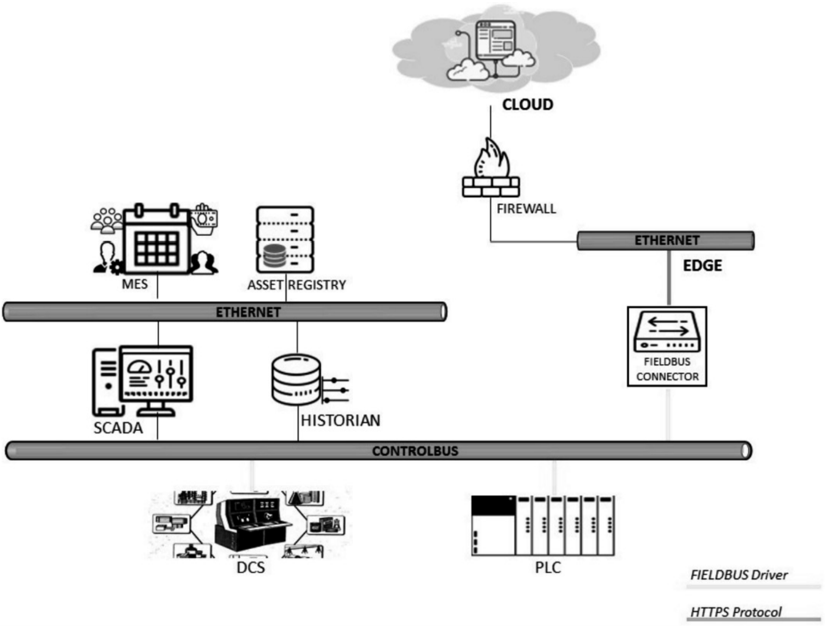

CLOUD
FIREWALL
ETHERNET
EDGE
MES
ASSET REGISTRY
ETHERNET
FIELDBUS CONNECTOR
SCADA
HISTORIAN
CONTROLBUS
DCS
PLC
FIELDBUS Driver
HTTPS Protocol

Figure 5.5 – The edge on a fieldbus

The most efficient way to secure this setup is to place the edge device in a DMZ, using two firewalls. The first one should control the interface facing the outside, and the second one should control the interface facing the CN, as shown in the following diagram:

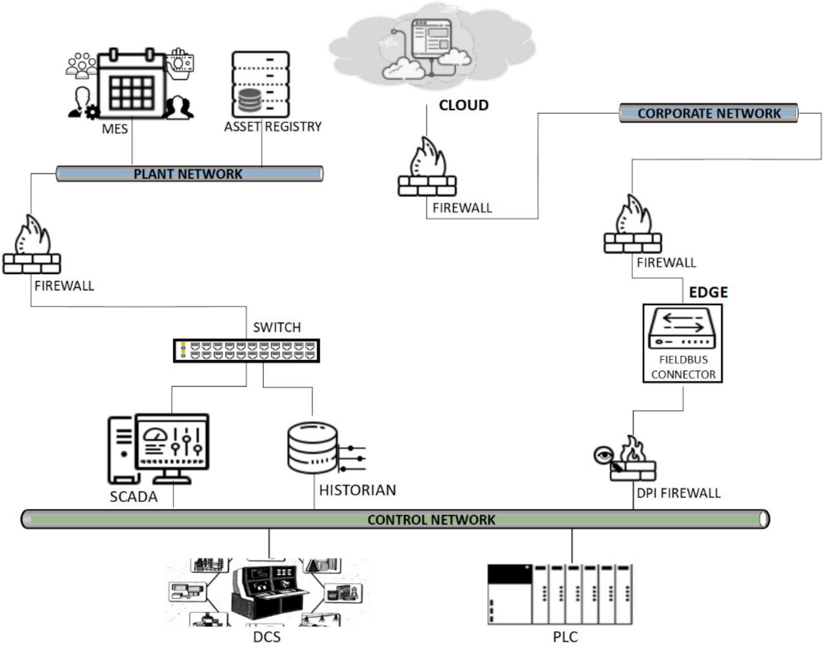

MES
ASSET REGISTRY
CLOUD
CORPORATE NETWORK
PLANT NETWORK
FIREWALL
FIREWALL
FIREWALL
EDGE
SWITCH
FIELDBUS CONNECTOR
SCADA
HISTORIAN
DPI FIREWALL
CONTROL NETWORK
DCS
PLC

Figure 5.6 – A secure edge on a fieldbus

To isolate the CN from the edge device, a DPI firewall must be inserted between the two. Since the DPI firewall provides deep-packet inspection at the application level, it can really understand the meaning of the packets passing through it, according to the specific control protocol. A typical use case is to secure a Modbus CN, allowing only reading and blocking any attempt at writing to the control devices linked to it. A DPI firewall for the Modbus protocol could be deployed, as described before, with rules that allow just read commands across the firewall and dropping all packets with invalid or unauthorized Modbus function codes. The other firewall that makes up the DMZ zone would manage the rules to allow the HTTPS traffic generated by the edge device facing the outside. There are two main difficulties associated with this setup:

- DPI firewall availability : A DPI firewall for managing and filtering the traffic of a specific industrial protocol may not be available, or it may be unable to implement all the rules needed to secure the required scenario
- Customer policy : Internal customer policies may not allow us to use the same device to pull data from the CN and push it simultaneously over the internet

Securing the edge on OPC DCOM

In this setup, the edge is not directly connected to the data source but, instead, to its OPC Classic server through an OPC Classic client, as shown in the following diagram:

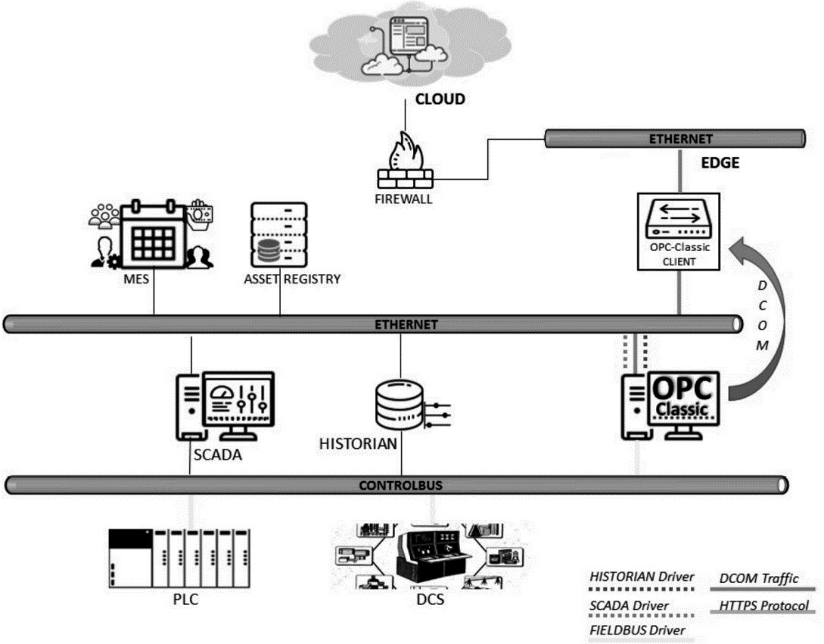

CLOUD
FIREWALL
ETHERNET
EDGE
OPC-Classic CLIENT
MES
ASSET REGISTRY
ETHERNET
D C O M
SCADA
HISTORIAN
OPC Classic
CONTROLBUS
PLC
DCS
HISTORIAN Driver
DCOM Traffic
SCADA Driver
HTTPS Protocol
FIELDBUS Driver

Figure 5.7 – The edge on OPC DCOM

The only way to secure this scenario is to place the edge device in a DMZ, using two firewalls, the first one controlling the interface facing the outside and the second one controlling the interface facing OPC Classic. This is shown in the following diagram:

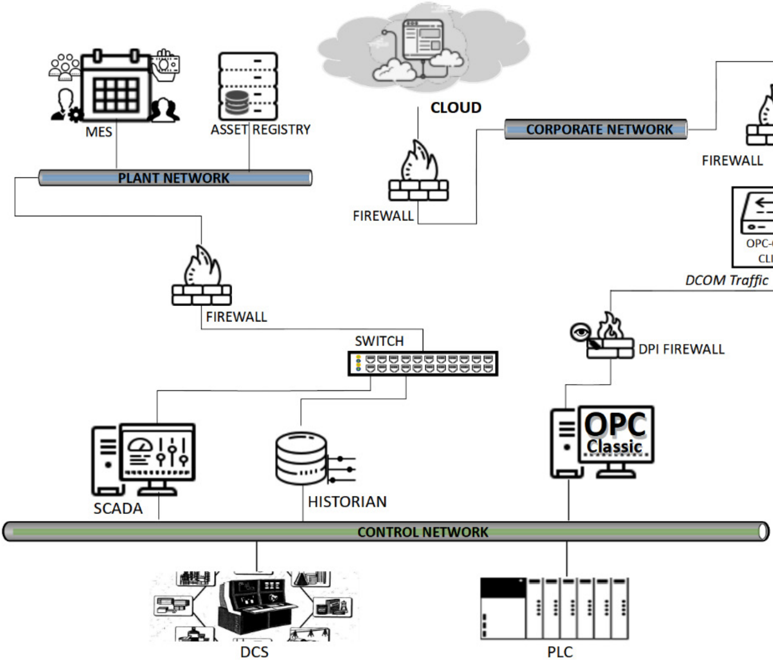

MES
ASSET REGISTRY
CLOUD
CORPORATE NETWORK
PLANT NETWORK
FIREWALL
FIREWALL
EDGE
OPC-Classic CLIENT
DCOM Traffic
FIREWALL
SWITCH
DPI FIREWALL
OPC Classic
SCADA
HISTORIAN
CONTROL NETWORK
DCS
PLC

Figure 5.8 – The secure edge on OPC DCOM

In this scenario, DCOM traffic occurs between the edge device and the OPC Classic device. The DCOM traffic is not as easy to manage using a firewall, since the DCOM communication is based on the opening of dynamic TCP ports. Since an OPC server might use any port (between `1024` and `65535`), any firewall placed between OPC Classic and the edge must necessarily allow the traffic through all those ports, making the firewall useless. Therefore, in order to keep the DCOM traffic under effective control, we have the following two options:

- Follow the Microsoft suggestion of limiting the range of port numbers that are dynamically allocated, modifying the settings of the Windows registries of the Windows box where the OPC Classic server runs. Unfortunately, this solution makes the configuration more complex for the system administrator, as every OPC host necessitates registry modifications in its Windows settings. Furthermore, subsequent testing has suggested that this technique is not fully applicable to some "OPC Classic" server products that don't work properly.
- Use the port number and protocol limitation provided by some OPC Classic implementations. Unfortunately, not all vendors of OPC products offer this option.

To isolate the edge device from OPC Classic effectively, a DPI firewall for OPC Classic must be used. Since the DPI firewall provides deep-packet inspection at the application level, it can really understand the meaning of the OPC Classic packets passing through it. The difficulties in managing this scenario are basically the same as those mentioned in the previous *Securing the edge on a fieldbus* section. We should also consider the constraints and limitations related to the use of DCOM communications. In general, the edge on OPC DCOM is not an easy setup to deal with, so it is advisable to introduce an OPC Proxy.

Securing the edge on OPC Proxy

In this setup, the internal OPC Proxy gathers data through DCOM from the OPC Classic server, tunneling it into a TCP channel that crosses the network boundary to reach the external OPC Proxy on the other side of the TCP channel. The external OPC Proxy exposes this data through its OPC UA internal server, making it available to the edge using its OPC UA client interface. This is shown in the following diagram:

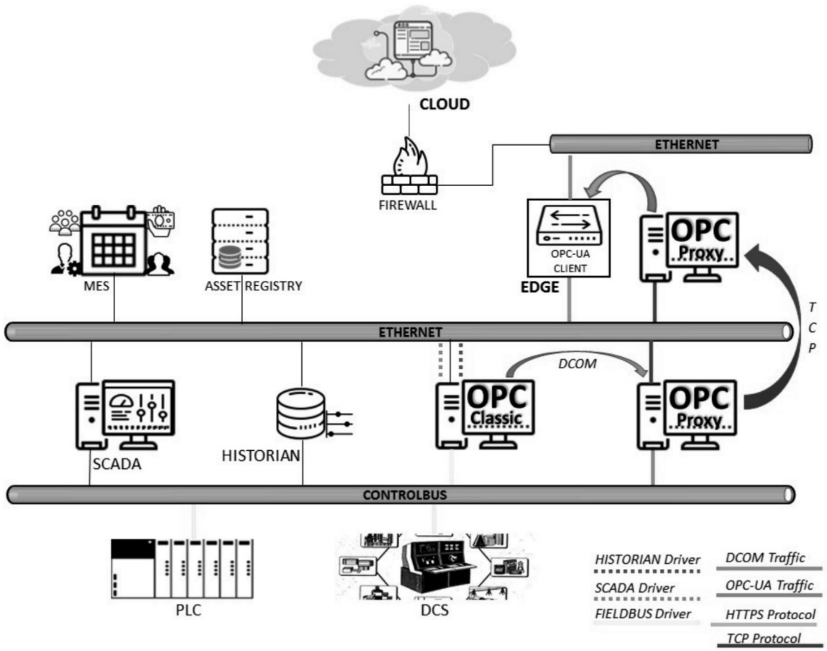

CLOUD
FIREWALL
ETHERNET
OPC-UA CLIENT
EDGE
OPC Proxy
MES
ASSET REGISTRY
ETHERNET
T C P
DCOM
OPC Classic
OPC Proxy
SCADA
HISTORIAN
CONTROLBUS
PLC
DCS
HISTORIAN Driver
SCADA Driver
FIELDBUS Driver
DCOM Traffic
OPC-UA Traffic
HTTPS Protocol
TCP Protocol

Figure 5.9 – The edge on OPC Proxy

This setup is much easier to secure from a networking perspective than the previous one. Since the DCOM traffic is tunneled through TCP using the OPC Proxy, two important goals from a security perspective are achieved:

- The edge does not interface with OPC Classic but, instead, with the OPC Proxy. This means there is no need to deal with the settings, configuration, and security of the DCOM. The lower boundary of the edge device toward the CN is the OPC UA internal server provided by the OPC Proxy.
- DCOM traffic occurs just on the internal Ethernet network between

> the internal OPC Proxy and the OPC Classic server. There is no DCOM traffic passing through firewalls, so there is no need to deal with the limitations of the TCP port opening or the Windows registry settings.

Then, the external OPC Proxy can be easily deployed in a DMZ using a three-way firewall, as shown in the following diagram. The three-way firewall links the external OPC Proxy to one of its interfaces and, using switches, connects the internal OPC Proxy to another interface and the edge device to a third one:

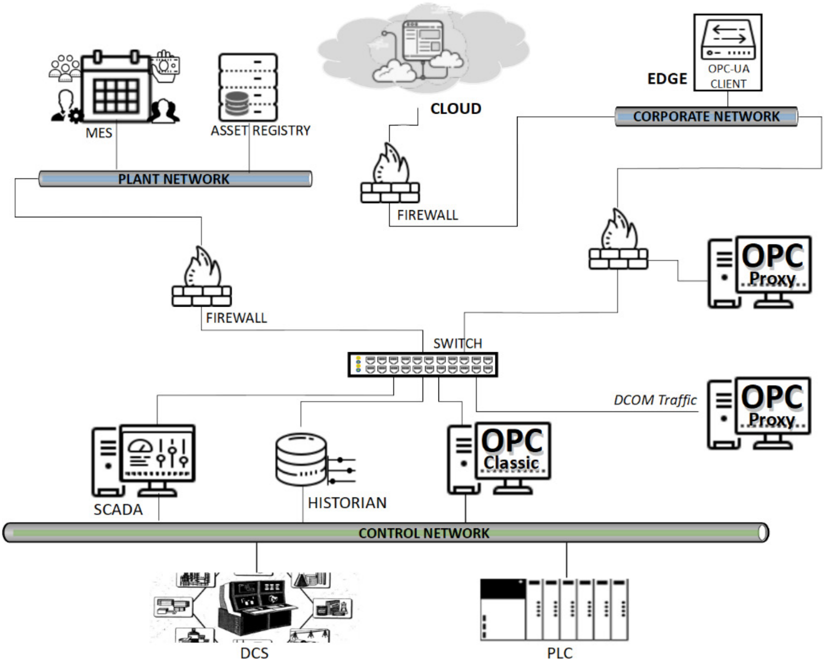

MES
ASSET REGISTRY
PLANT NETWORK
CLOUD
EDGE
OPC-UA CLIENT
CORPORATE NETWORK
FIREWALL
FIREWALL
OPC Proxy
SWITCH
DCOM Traffic
OPC Proxy
SCADA
HISTORIAN
OPC Classic
CONTROL NETWORK
DCS
PLC

Figure 5.10 – Secure edge on OPC Proxy

In this setup, by placing the external OPC Proxy in a DMZ and tunneling the DCOM traffic through TCP, three important goals are achieved from a security perspective:

- The OPC Classic server does not receive any requests coming from devices outside its network. All requests to the OPC Classic server pass through the internal OPC Proxy linked to the same Ethernet network.
- There is no need to move DCOM traffic across the network boundaries. DCOM traffic only occurs between the OPC Classic server and the internal OPC Proxy, which are linked to the same Ethernet network.
- The external OPC Proxy is a DMZ. This means that no request comes directly from the outside to any device on the CN.

Securing the edge on OPC UA

In the edge on OPC UA setup, the edge is directly connected to the OPC UA server using its OPC UA client interface, as shown in the following diagram:

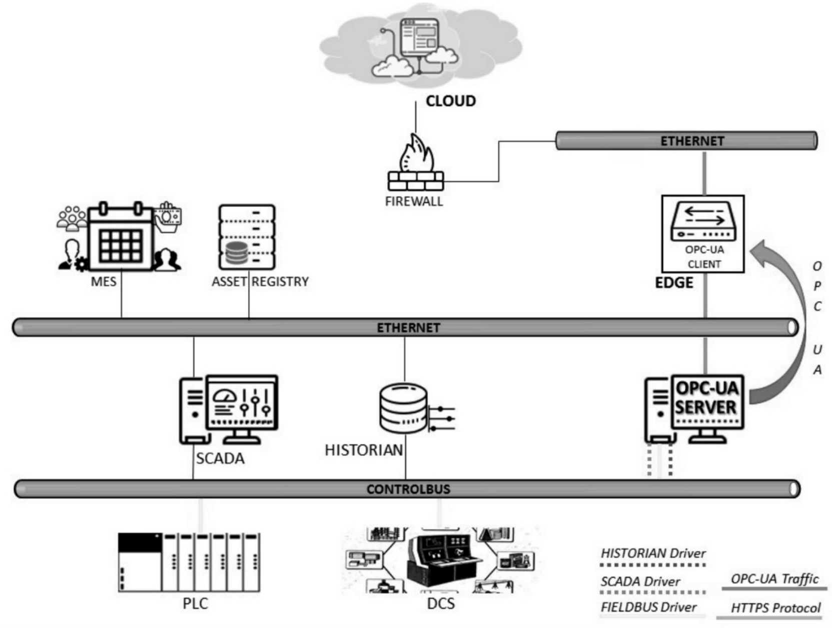

CLOUD
FIREWALL
ETHERNET
OPC-UA CLIENT
EDGE
MES
ASSET REGISTRY
ETHERNET
O P C U A
SCADA
HISTORIAN
OPC-UA SERVER
CONTROLBUS
PLC
DCS
HISTORIAN Driver
SCADA Driver
FIELDBUS Driver
OPC-UA Traffic
HTTPS Protocol

Figure 5.11 – The edge on OPC UA

This setup uses the security model provided by the OPC UA, which is firewall-friendly. It needs a firewall to filter and secure the HTTPS/TCP traffic due to the requests coming from the edge device, placed outside of the network where the OPC UA server is linked. The deployment of another firewall between the edge and the network linked to the internet creates a DMZ for the edge deployment, as shown in the following diagram:

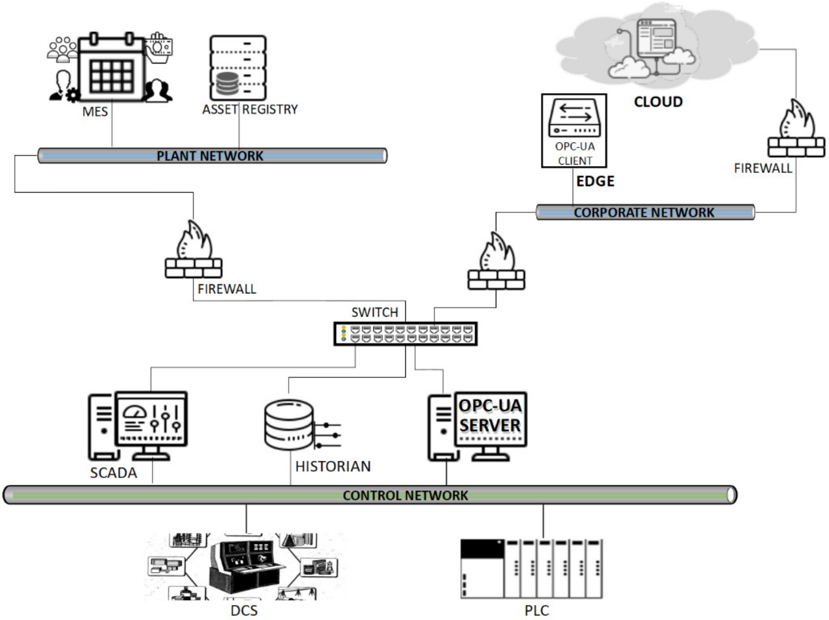

MES
ASSET REGISTRY
PLANT NETWORK
CLOUD
OPC-UA
CLIENT
EDGE
FIREWALL
CORPORATE NETWORK
FIREWALL
SWITCH
SCADA
HISTORIAN
OPC-UA
SERVER
CONTROL NETWORK
DCS
PLC

Figure 5.12 – The secure edge on OPC UA

The OPC UA has been designed for devices with very different computational capabilities. According to the OPC UA standards, a device cannot fully implement all the features of the OPC UA standards. The implementation level of OPC UA is delegated by the vendor. This setup requires a risk analysis to understand whether, and how much, the implementation of the OPC UA security model provided by its vendor fits the security requirements of the specific scenario. Eventually, an additional

DPI firewall could be deployed between the OPC UA server and the CN to further segregate the OPC UA server.

Securing OPC UA on a controller

In this scenario, the OPC UA server for a PLC or DCS runs on a specific board, which can be installed in the rack of the controller itself. The edge is connected to the OPC UA server, which is hosted by the controller through the OPC UA client interface, as shown in the following diagram:

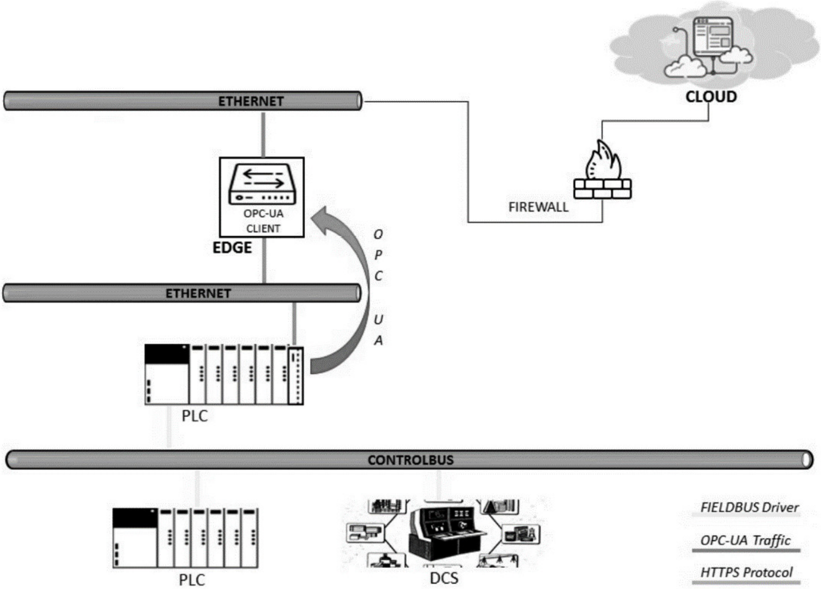

ETHERNET
CLOUD
OPC-UA
CLIENT
EDGE
FIREWALL
O
P
C
U
A
ETHERNET
PLC
CONTROLBUS
PLC
DCS
FIELDBUS Driver
OPC-UA Traffic
HTTPS Protocol

Figure 5.13 – OPC UA on a controller

This setup, like the previous one, entirely uses OPC UA interfaces. In this scenario, a firewall must be placed to filter and secure the HTTPS/TCP traffic due to the requests coming from the edge device, placed outside of the network where the OPC UA server is linked. The deployment of another firewall between the edge and the network linked to the internet creates a DMZ for the edge deployment, as shown in the following diagram:

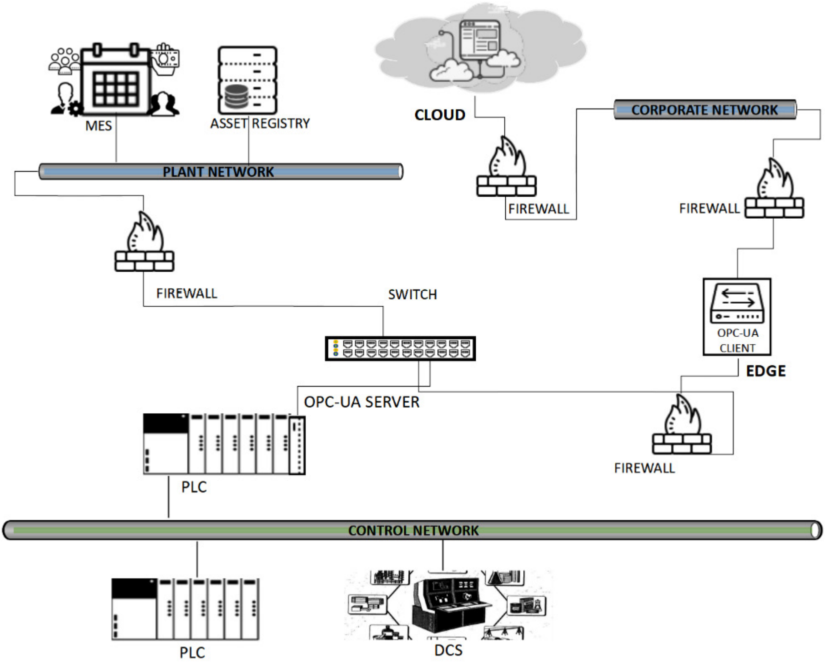

MES
ASSET REGISTRY
CLOUD
CORPORATE NETWORK
PLANT NETWORK
FIREWALL
FIREWALL
FIREWALL
SWITCH
OPC-UA CLIENT
EDGE
OPC-UA SERVER
FIREWALL
PLC
CONTROL NETWORK
PLC
DCS

Figure 5.14 – Secure OPC UA on a controller

Unfortunately, in this setup, there is no option to deploy a DPI firewall to segregate the OPC UA server further from the CN, since the communication between the OPC UA server and the controller occurs through the internal bus of the controller itself. There are two additional considerations that should be taken into account regarding this setup:

- In general, when the OPC UA server is hosted on an integrated board of the controller, it runs on a Linux box, which is a bit more resilient from a cybersecurity perspective than a Windows box

- A software firewall can be installed on the Linux box in which the OPC UA server runs. It can be configured to further segregate the CN to which the OPC UA server is linked via the controller bus

In any case, since this setup exclusively uses the security model provided by OPC UA, a careful analysis of the OPC UA security model implemented by the vendor must be carried out, ensuring that it can cover the security requirements of the specific use case.

Summary

In this chapter, we outlined the DiD approach. You learned that the goal of a DiD strategy is the achievement of a security posture through the coordinated and combined use of multiple security countermeasures, based on two main concepts – defense in multiple places and layered defenses. We looked at how DiD is based on the integration of three different elements – people, technology, and operating methods. Since firewalls are an important part of securing a CN, we also provided a short description of the different firewall classes. After that, we explored the most common architectures to secure the industrial devices linked to the CN.

Following this, we looked at how to segregate a CN using DMZ and VLAN. We examined the most common security practices currently used in the industrial control environment, and we analyzed the five options to connect the edge to industrial data sources from a cybersecurity point of view. We also looked at how to secure these from a networking perspective, according to the ICS standards and the related best practices.

In the next chapter, we will discover how to implement a basic data flow with OPC UA and Node-RED.

6

Performing an Exercise Based on Industrial Protocols and Standards

In *Chapter 4* , we learned about the Open Platform Communications (OPC) protocol. We looked at how the features of its original implementation were based on the Microsoft COM/DCOM architecture. We also looked at how it evolved into its current Unified Architecture (UA) and how it uses open and interoperable standards to overcome constraints and security issues. After that, we learned how to gather industrial data from different sources in different deployment scenarios, using the edge gateway and OPC UA. Finally, we briefly discussed the differences between the edge gateway and edge computation.

In this chapter, we will look at how to implement a basic data flow with OPC UA and Node-RED. We will carry out a practical example using the following:

- The OPC UA Simulation Server
- The edge

Technical requirements

In this chapter, we will need the following prerequisites:

- Git : https://git-scm.com/downloads
- Node.js : https://nodejs.org

The OPC UA Simulation Server

In this section, we will simulate a very simple industrial data flow to implement data acquisition through a simple edge device. To simulate the industrial data flow, we will use an OPC UA

Simulation Server to generate the signals. OPC UA is managed by the OPC Foundation (https://opcfoundation.org/) and is the most common industrial operability standard. Several vendors have implemented their own solutions according to the OPC UA standard. We can find other open source drivers for the most common languages, such as Node.js, Python, and Java.

OPC UA Server Node.js

The OPC UA implementation for Node.js (http://node-opcua.github.io/) is one of the most popular libraries. It supports both the client and the server. In this exercise, we will use the library indirectly when we work with Node-RED. We will also reuse it when we work with AWS and Azure.

Starting an OPC UA sample server

The easiest way to start a simple OPC UA server is to use the Node.js implementation. From the command line, we can run the following commands:

```
$ git clone https://github.com/
PacktPublishing/Hands-On-Industrial-
Internet-of-Things-Second-Edition.git
$ cd Hands-On-Industrial-Internet-of-
Things-Second-Edition
$ cd Chapter6
$ npm install
$ npm start
```

This server publishes three simple random measures called **`Temperature`** , **`Percentage`** , and **`MyVariable2`** . We will use this simple server for the exercises in this book.

Adding a new custom signal

We should add a new signal for editing the **`server.js`** file. Open **`server.js`** , located on **`node-opcua-sampleserver`** , and from *line 88* , add the following code:

```
  const nodeVariable5 =
  namespace.addVariable({
        componentOf: myDevice,
        nodeId: "s=CPU",
        browseName: "Percentage CPU
Used",
        dataType: "Double",
        minimumSamplingInterval: 1000,
        value: {
            get: () => {
const previousUsage =
process.cpuUsage();
              const startDate =
Date.now();
              while (Date.now() -
startDate < 500);
              const usage =
process.cpuUsage(previousUsage);
              const value = 100 *
(usage.user + usage.system) /
((Date.now() - startDate) * 1000);
                return new Variant({
dataType: DataType.Double, value:
value });
            }
        }
    });
```

Restart the command:

server with the following command:

```
$ npm start
```

> ***Note***

> *The* `server_var_cpu.js` *file in the official repository has been already changed to support this new variable. You can replace* `server.js` *with* `server_var_cpu.js` *and start NodeJS.*

Prosys OPC UA Simulator – optional step

The Prosys OPC UA server is one of the most common OPC UA servers for Windows. We can use the Prosys OPC UA Simulation Server to simulate multiple measures. This step is optional because, in this book, we are going to use the simulation server developed in the previous section. If you want to use a more powerful server, the Prosys server is the best-in-class server and we recommend familiarizing yourself with this simulation server.

Installing the Prosys server

To start the Prosys OPC server, we need to download the package from https://prosysopc.com/products/opc-ua-simulation-server/ .

Subscribe to the Prosys' agreements and you will receive a link to download it (you need to use a company email address).

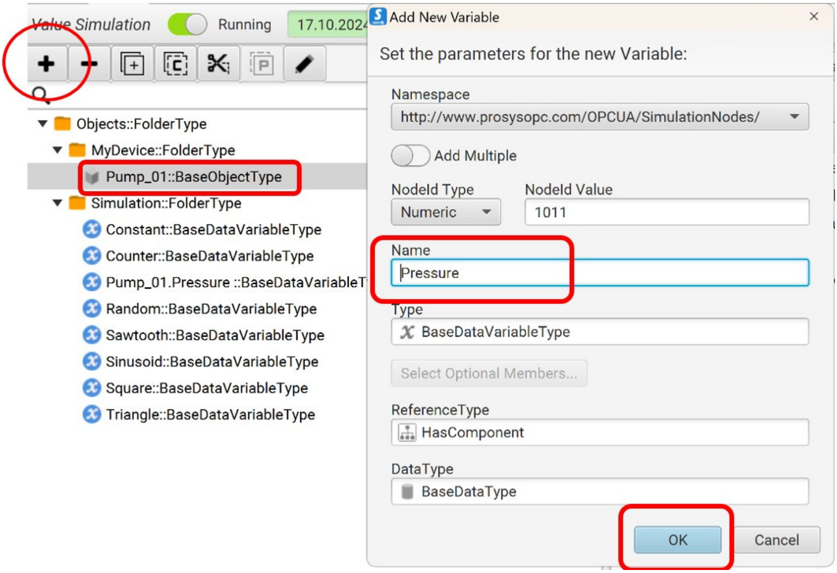
All we need to do to install the Prosys server is click on the installation package and accept all steps.

Simulating measures

Since we don't have a real industrial data source, we have to simulate the measures. To build a simulated measure, follow these steps:

1. Open the Prosys OPC UA Simulation Server user interface.
2. Click on the Objects tab.
3. Click on the + button, and add a new folder called `MyDevice` .
4. Then add a new object called `Pump_01` .
5. Add a new variable called `Pressure` . Then, click on the OK button.

The preceding steps are shown in the following screenshot:



Objects::FolderType
MyDevice::FolderType
Pump_01::BaseObjectType
Simulation::FolderType
Constant::BaseDataVariableType
Counter::BaseDataVariableType
Pump_01.Pressure ::BaseDataVariableT
Random::BaseDataVariableType
Sawtooth::BaseDataVariableType
Sinusoid::BaseDataVariableType
Square::BaseDataVariableType
Triangle::BaseDataVariableType

Set the parameters for the new Variable:

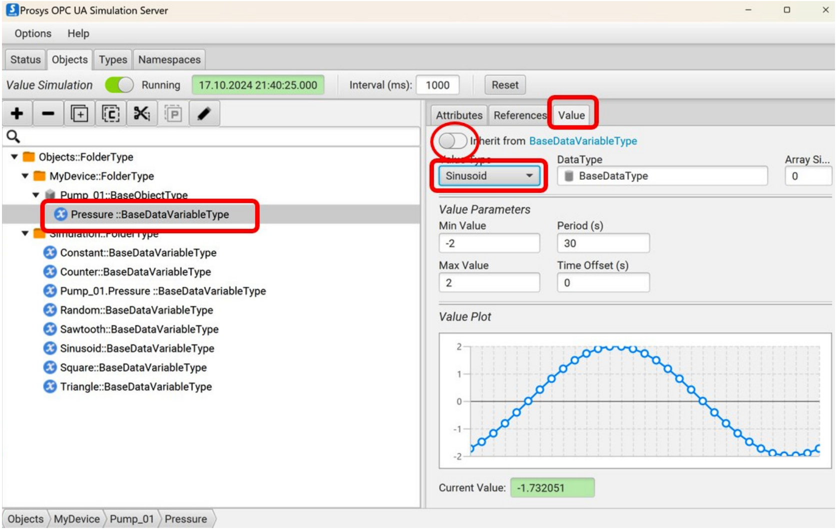
Namespace: http://www.prosysopc.com/OPCUA/SimulationNodes/
NodeId Type: Numeric
NodeId Value: 1011
Name: Pressure
Type: BaseDataVariableType
ReferenceType: HasComponent
DataType: BaseDataType

Figure 6.1 – Defining simulated measure

After creating this simulated measure, we can define the simulation mechanism.

1. Click on the Value tab.
2. Disable Inherit from BasicDataVaribleType .
3. Finally, select the Sinusoid method from the drop-down list.

The preceding steps are shown in the following screenshot:


Prosys OPC UA Simulation Server
Options
Help
Status
Objects
Types
Namespaces
Reset
Attributes
References
Value
Inherit from BaseDataVariableType
Objects::FolderType
MyDevice::FolderType
Pump_01::BaseObjectType
Pressure ::BaseDataVariableType
Simulation::FolderType
Constant::BaseDataVariableType
Counter::BaseDataVariableType
Pump_01.Pressure ::BaseDataVariableType
Random::BaseDataVariableType
Sawtooth::BaseDataVariableType
Sinusoid::BaseDataVariableType
Square::BaseDataVariableType
Triangle::BaseDataVariableType
Value Parameters
Value Plot
2
1
0
-1
-2
Objects
MyDevice
Pump_01
Pressure

Value Simulation: Running 17.10.2024 21:40:25.000
Interval (ms): 1000
Value Type: Sinusoid
DataType: BaseDataType
Array Si...: 0
Min Value: -2
Period (s): 30
Max Value: 2
Time Offset (s): 0
Current Value: -1.732051

Figure 6.2 – Setting the simulated signal

Now, you are ready to connect an edge to this simulation server. Let's begin!

The edge

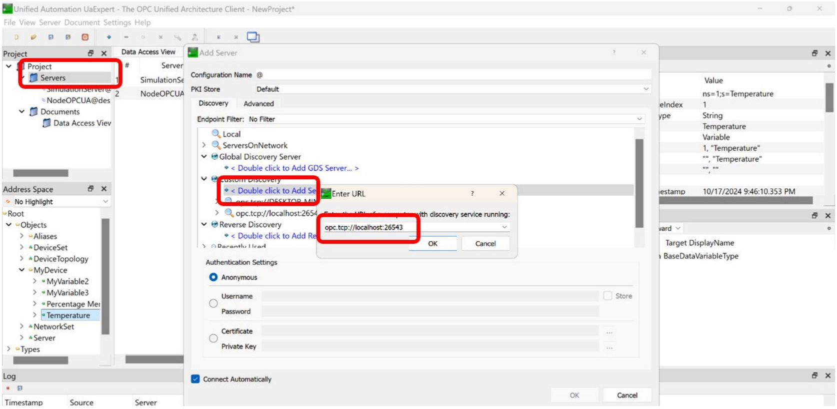
Let's start with our first IIoT component: the edge. The edge is a key component able to connect to the data source and send it to the cloud. In other words, the edge is the bridge between the data source and the central IoT system.

Installing a simple edge client

Before starting our edge, we suggest installing a simple graphical client. This client will allow us to explore in a more graphical way the OPC UA server, identify the registry and, in some cases, debug the connectivity. From https://www.unified-automation.com/products/development-tools/uaexpert.html , install the `uaexpert` client.

Now, we can configure the server and we can add the OPC UA Simulation Server. Navigate to Servers | Add new | Customer Discovery | Add new | `opc.tcp://localhost:26543` .

The following screenshot shows the steps to accomplish this:



Unified Automation UaExpert - The OPC Unified Architecture Client - NewProject*

Add Server

Configuration Name: @

PKI Store: Default

Endpoint Filter: No Filter

Enter URL

opc.tcp://localhost:26543

Authentication Settings: ● Anonymous

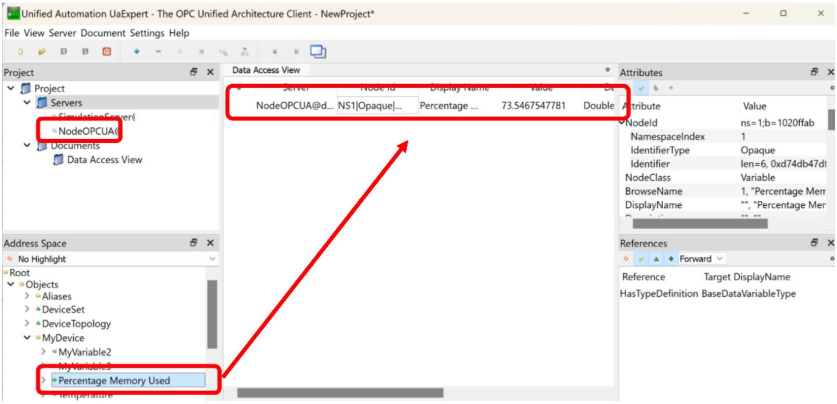
☑ Connect Automatically

File View Server Document Settings Help
Project
Project
Servers
NodeOPCUA@des
Documents
Data Access Viev
Data Access View
#
Server
1 SimulationSe
2 NodeOPCUA
Discovery
Advanced
Local
ServersOnNetwork
Global Discovery Server
< Double click to Add GDS Server... >
< Double click to Add Se
opc.tcp://localhost:2654
Reverse Discovery
< Double click to Add Re
ith discovery service running:
OK
Cancel
Username
Password
Store
Certificate
Private Key
OK
Cancel
Address Space
No Highlight
Root
Objects
Aliases
DeviceSet
DeviceTopology
MyDevice
MyVariable2
MyVariable3
Percentage Mer
Temperature
NetworkSet
Server
Types
Log
Timestamp
Source
Server
Value
ns=1;s=Temperature
1
String
Temperature
Variable
1, "Temperature"
"", "Temperature"
10/17/2024 9:46:10.353 PM
Target DisplayName
BaseDataVariableType

Figure 6.3 – Step 1 – configuring the server

Accept the certificate, click on None , and start your exploration.

From the left panel, drag and drop the Percentage memory used node onto the main panel. The following screenshot shows the steps to accomplish this:


Unified Automation UaExpert - The OPC Unified Architecture Client - NewProject*
File View Server Document Settings Help
Project
Project
Servers
SimulationServer@
NodeOPCUA@
Documents
Data Access View
Address Space
No Highlight
Root
Objects
Aliases
DeviceSet
DeviceTopology
MyDevice
MyVariable2
MyVariable3
Percentage Memory Used
Temperature
Data Access View
Attributes
References
Forward

#	Server	Node Id	Display Name	Value	Da
	NodeOPCUA@d...	NS1|Opaque|...	Percentage ...	73.5467547781	Double

Attribute	Value
NodeId	ns=1;b=1020ffab
NamespaceIndex	1
IdentifierType	Opaque
Identifier	len=6, 0xd74db47d
NodeClass	Variable
BrowseName	1, "Percentage Mem
DisplayName	"", "Percentage Mer

Reference	Target DisplayName
HasTypeDefinition	BaseDataVariableType

Figure 6.4 – Step 2 – showing the memory used

You can also add several servers and measures by following the procedure shown earlier.

We have tested that our OPC UA Simulation Server works.

The Node-RED edge

We can develop our edge with different technologies and frameworks. In *Chapter 10* , *Implementing a Cloud Industrial IoT Solution with AWS* , and *Chapter 12* , we will learn about the AWS edge (Greengrass) and Azure Edge. We will also develop a simple edge MQTT with the AWS SDK and Azure SDK. To develop a simple edge to access the OPC UA server, we can use Node.js and the OPC UA client (http://node-opcua.github.io/). In this simple exercise, we are going to use a graphical interface called Node-RED .

Node-RED

Node-RED is an open source independent edge device, based on Node.js. Follow these steps to install Node-RED:

1. Assuming Node.js is already installed, we can install Node-RED from the command console with the following command:

```
$ npm install -g --unsafe-perm node-red
```

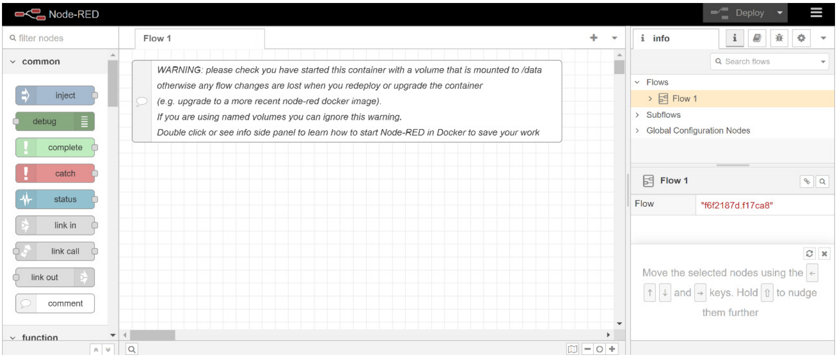
2. In the same directory, we need to install the support for OPC UA, as follows:

```
$ npm install -g node-red-contrib-opcua
```

3. We are now ready to work with Node-RED. From the command console, enter the following command:

```
$ node-red
```

We can then connect to `http://127.0.0.1:1880` to see the user interface with the OPC UA modules installed, as shown in the following screenshot:


Node-RED
Deploy
filter nodes
common
inject
debug
complete
catch
status
link in
link call
link out
comment
function
Flow 1
info
Search flows
Flows
Flow 1
Subflows
Global Configuration Nodes
Flow 1

WARNING: please check you have started this container with a volume that is mounted to /data otherwise any flow changes are lost when you redeploy or upgrade the container (e.g. upgrade to a more recent node-red docker image).
If you are using named volumes you can ignore this warning.
Double click or see info side panel to learn how to start Node-RED in Docker to save your work

Flow "f6f2187d.f17ca8"

Move the selected nodes using the ← ↑ ↓ and → keys. Hold ⇧ to nudge them further

Figure 6.5 – Node-RED UI

Node-RED on Docker

You can also run the `node-red` command from Docker (see *Chapter 7* , for more information). From the command line, enter the following command:

```
$ mkdir data
$ docker run -it -p 1880:1880 -v data:/
data --name mynodered nodered/ node-red
```

On a separate command line, run the following:

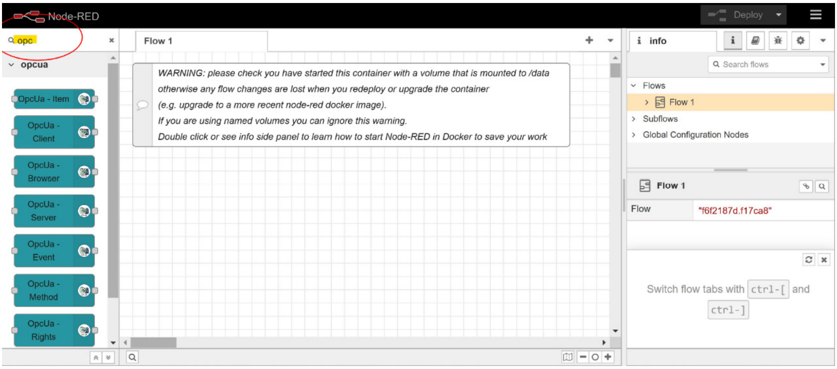
```
$ docker exec -it mynodered npm install
node-red-contrib-opcua
$ docker commit mynodered nodered/node-
red-opc
```

`docker commit` allows us to save the changes made to the running container on a new image so that we will not lose them if we restart the container.

Close the running Docker container and start a new Docker container with a new image:

```
$ docker run -it -p 1880:1880 -v data:/
data --name mynodered nodered/ node-red-
opc
```

Now, you can connect to Node-RED on `http://localhost:1880/` and see that the OPC UA plugin is installed:


Node-RED
Deploy
opc
opcua
OpcUa - Item
OpcUa - Client
OpcUa - Browser
OpcUa - Server
OpcUa - Event
OpcUa - Method
OpcUa - Rights
Flow 1
info
Search flows
Flows
Flow 1
Subflows
Global Configuration Nodes
Flow 1

WARNING: please check you have started this container with a volume that is mounted to /data otherwise any flow changes are lost when you redeploy or upgrade the container (e.g. upgrade to a more recent node-red docker image). If you are using named volumes you can ignore this warning. Double click or see info side panel to learn how to start Node-RED in Docker to save your work

Flow "f6f2187d.f17ca8"

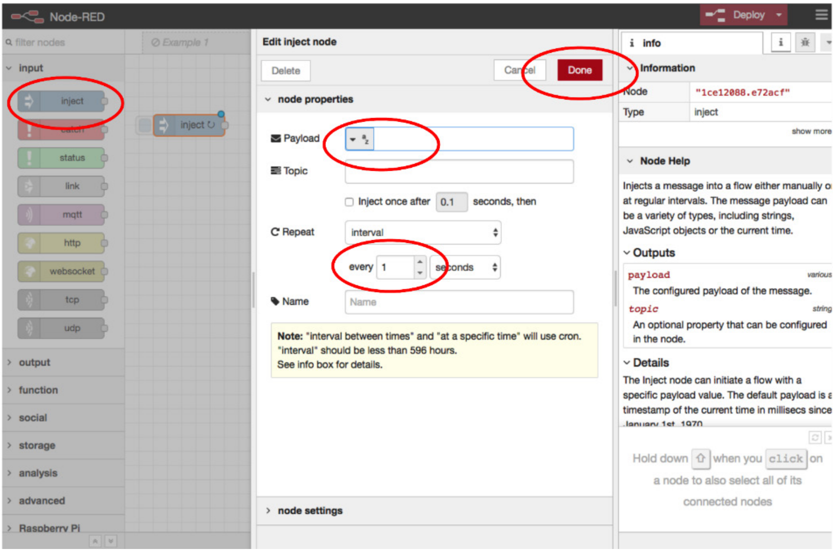
Switch flow tabs with `ctrl-[` and `ctrl-]`

Figure 6.6 – Node-RED web interface with OPC support

Building the flow

We can now build our flow to access the OPC UA server. Follow these steps:

From the input on the left, we have to drag and drop the inject node into the Flow 1 area. In the Payload field, select string and set the Repeat field to every `1` seconds. The following screenshot shows the steps to accomplish this:


Node-RED
Deploy
filter nodes
input
inject
catch
status
link
mqtt
http
websocket
tcp
udp
output
function
social
storage
analysis
advanced
Raspberry Pi
Example 1
inject
Delete
Cancel
Done
node settings
info
show more

Edit inject node

node properties

Payload

Topic

☐ Inject once after 0.1 seconds, then

Repeat: interval

every 1 seconds

Name: Name

Note: "interval between times" and "at a specific time" will use cron. "interval" should be less than 596 hours. See info box for details.

Information

Node	"1ce12088.e72acf"
Type	inject

Node Help

Injects a message into a flow either manually o at regular intervals. The message payload can be a variety of types, including strings, JavaScript objects or the current time.

Outputs

payload *various*
The configured payload of the message.

topic *string*
An optional property that can be configured in the node.

Details

The Inject node can initiate a flow with a specific payload value. The default payload is a timestamp of the current time in millisecs since January 1st, 1970

Hold down ⇧ when you click on a node to also select all of its connected nodes

Figure 6.7 – Step 1 – configuring the flow

We can configure the access to OPC UA using the `OpcUa Client` and the `OpcUa Item` sections. We need to drag and drop the two nodes into the Flow 1 area and connect them to each other, as depicted in the following screenshot:

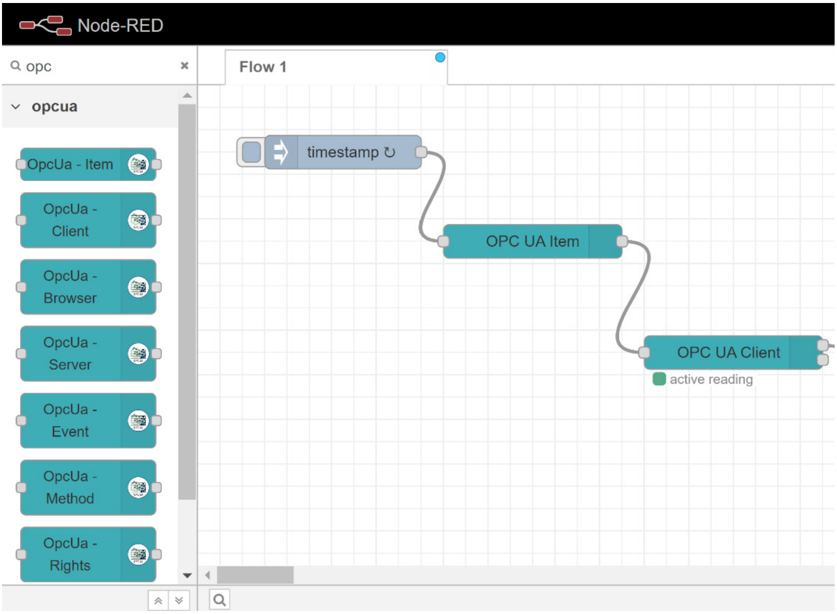
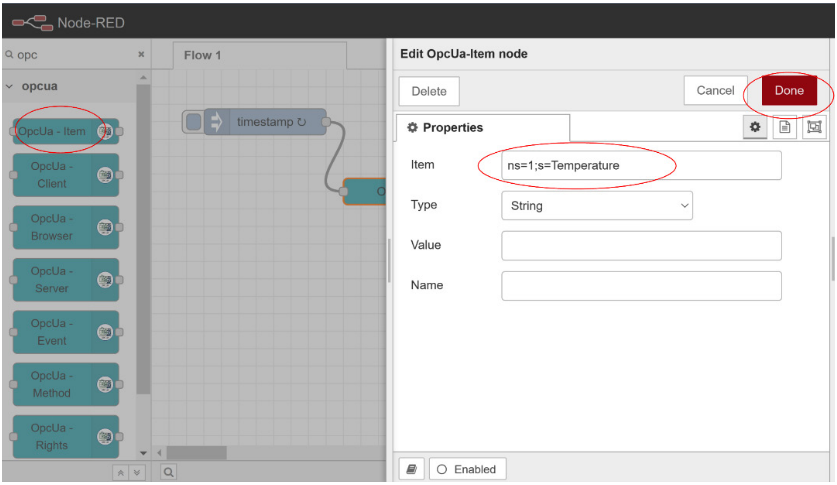

Node-RED
opc
opcua
OpcUa - Item
OpcUa - Client
OpcUa - Browser
OpcUa - Server
OpcUa - Event
OpcUa - Method
OpcUa - Rights
Flow 1
timestamp ↻
OPC UA Item
OPC UA Client
active reading

Figure 6.8 – Step 2 – configuring the OPC UA

Double-click on `OpcUa Item` to enable the configuration page. In the `Item` field, write the following to indicate which measure to get from the OPC UA Server:

```
ns=1;s=Temperature
```

The following screenshot shows how to do this:


Node-RED
opc
opcua
OpcUa - Item
OpcUa - Client
OpcUa - Browser
OpcUa - Server
OpcUa - Event
OpcUa - Method
OpcUa - Rights
Flow 1
timestamp ↻
Delete
Cancel
Done
Properties
Enabled

Edit OpcUa-Item node

Item: ns=1;s=Temperature

Type: String

Value:

Name:

Figure 6.9 – Step 3 – configuring the OPC UA item

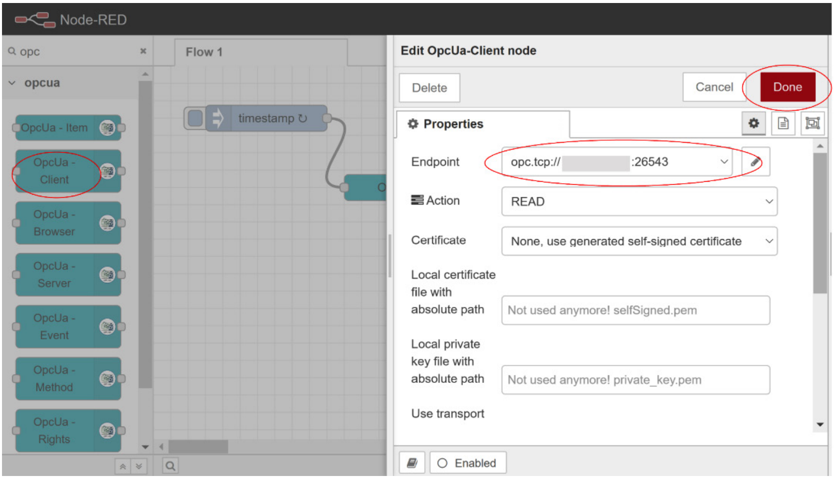
On the OPC UA client, we can configure the following endpoint:

```
opc.tcp://localhost:26543
```

If you are using docker image you need to replace localhost with ip local address, for instance:

```
opc.tcp://192.168.0.1:26543
```

The following screenshot shows the previous step:


Node-RED
opc
opcua
OpcUa - Item
OpcUa - Client
OpcUa - Browser
OpcUa - Server
OpcUa - Event
OpcUa - Method
OpcUa - Rights
Flow 1
timestamp ↻
Edit OpcUa-Client node
Delete
Cancel
Done
Properties
○ Enabled

Endpoint: opc.tcp://[redacted]:26543

Action: READ

Certificate: None, use generated self-signed certificate

Local certificate file with absolute path: Not used anymore! selfSigned.pem

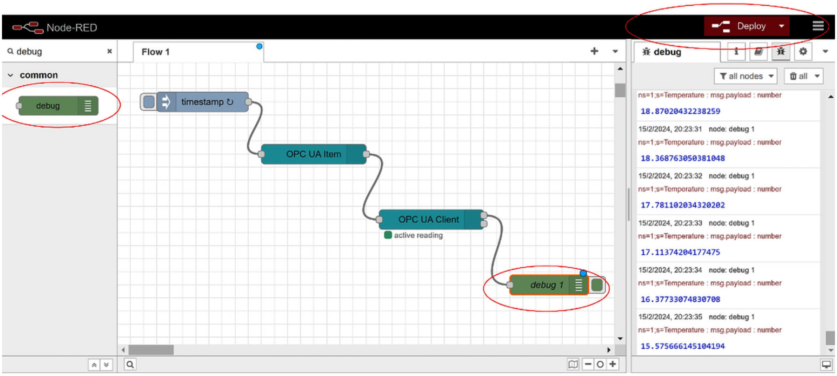
Local private key file with absolute path: Not used anymore! private_key.pem

Use transport

Figure 6.10 – Step 4 – configuring the OPC UA connection

To complete our flow, we need to connect the debug node as the output.

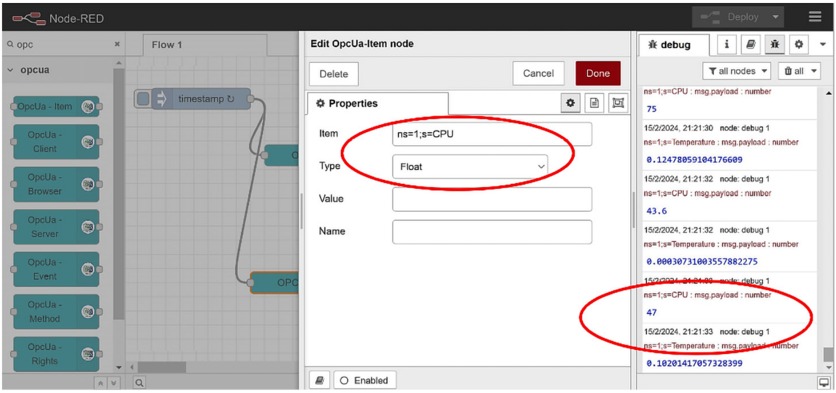
Finally, we need to click on the Deploy dropdown to start our flow. The final flow is shown in the following screenshot:


Node-RED
Deploy
debug
common
debug
Flow 1
timestamp ↻
OPC UA Item
OPC UA Client
active reading
debug 1
debug
all nodes
all

```
ns=1;s=Temperature : msg.payload : number
18.87020432238259
15/2/2024, 20:23:31   node: debug 1
ns=1;s=Temperature : msg.payload : number
18.368763050381048
15/2/2024, 20:23:32   node: debug 1
ns=1;s=Temperature : msg.payload : number
17.781102034320202
15/2/2024, 20:23:33   node: debug 1
ns=1;s=Temperature : msg.payload : number
17.11374204177475
15/2/2024, 20:23:34   node: debug 1
ns=1;s=Temperature : msg.payload : number
16.37733074830708
15/2/2024, 20:23:35   node: debug 1
ns=1;s=Temperature : msg.payload : number
15.575666145104194
```

Figure 6.11 – Node-RED final flow

In the Debug tab, we can see the simulated measures. Repeat *Steps 2* to *4* and also add the **`Percentage CPU`** measure added in the *Adding a new custom signal* section. The final flow is shown in the following screenshot:


Node-RED
Deploy
opc
opcua
OpcUa - Item
OpcUa - Client
OpcUa - Browser
OpcUa - Server
OpcUa - Event
OpcUa - Method
OpcUa - Rights
Flow 1
timestamp ↻
OPC
Edit OpcUa-Item node
Delete
Cancel
Done
Properties
Enabled
debug
all nodes
all

Item: ns=1;s=CPU
Type: Float
Value:
Name:

```
ns=1;s=CPU : msg.payload : number
75
15/2/2024, 21:21:30   node: debug 1
ns=1;s=Temperature : msg.payload : number
0.12478059104176609
15/2/2024, 21:21:32   node: debug 1
ns=1;s=CPU : msg.payload : number
43.6
15/2/2024, 21:21:32   node: debug 1
ns=1;s=Temperature : msg.payload : number
0.00030731003557882275
15/2/2024, 21:21:[illegible]   node: debug 1
ns=1;s=CPU : msg.payload : number
47
15/2/2024, 21:21:33   node: debug 1
ns=1;s=Temperature : msg.payload : number
0.10201417057328399
```

Figure 6.12 – Adding the custom measure

We can use Node-RED to send data to the IIoT middleware using the MQTT node.

Securing your OPC UA Server

> *We would like to note that we have configured the OPC UA server and client without authentication; this policy is strongly NOT recommended (see*

> *Chapter 5). The purpose of this exercise is just to learn more about OPC UA, edge, and servers.*

Summary

In this chapter, we have implemented our first data flow. We will reuse the Prosys OPC UA Simulation Server in two subsequent chapters – *Chapter 10* and *Chapter 12* . In these, we will implement a more practical data flow with Azure and AWS.

In the next chapter, we will look at IIoT middleware and understand the basic components of a central IIoT system. We will learn about its basic functionalities and some key use cases that are useful in an industrial context.

Part 2:Industrial IoT Architecture

This part introduces the concepts of platform as a service, microservices, and containers. The reader will be guided through a step-by-step process to develop a comprehensive wing-to-wing IoT architecture. Three distinct architectures will be implemented: one utilizing open source technologies, another based on AWS, and the third leveraging Azure.

This part includes the following chapters:

- *Chapter 7 , Developing Industrial IoT and Architecture*
- *Chapter 8 , Implementing a Custom Industrial IoT Platform*
- *Chapter 9 , Building an AWS Industrial IOT Solution*
- *Chapter 10 , Implementing an Industrial IOT Data Flow with AWS*
- *Chapter 11 , Performing a Practical Industrial IoT Solution with Azure*
- *Chapter 12 , Implementing an Industrial IoT Data Flow with Azure*

7

Developing Industrial IoT and Architecture

In this chapter, we will learn about the basic technologies required to develop an IIoT architecture. We will focus on the most active parts (asset registry, data storage, and analytics). We will also look at different use cases and how these affect your choice of technology.

By the end of this chapter, you will have learned the basic concepts of how to develop an IIoT architecture.

We will cover the following topics:

- The IIoT platform and architectures
- Microservices, containers, and serverless computing

- The standard IIoT flow
- Understanding the time-series technologies
- Asset registry
- Data processing and the analytics platform
- Hands-on with InfluxDB, NODE-red, and OPC UA
- Using Docker Compose and Docker Build

Technical requirements

This chapter will require the following software to be installed on your local PC:

- Docker Community Edition: https://www.docker.com/community-edition
- Git: https://git-scm.com/downloads

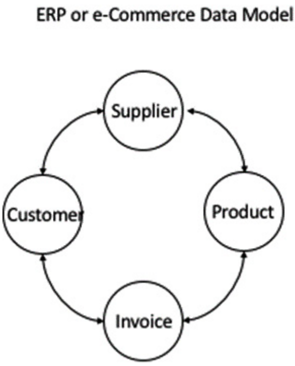
Introduction to the IIoT platform and architectures

From an architectural point of view, an IoT platform presents significant challenges:

- The amount of data acquired, ingested, and processed requires high bandwidth, storage, and computational capabilities

- The devices are distributed across a vast geographical area
- Businesses require their architecture to be evolutionary so that new services and capabilities can be added daily to deploy to customers

More so than other standard applications, flexibility and scalability are vital to a typical IoT platform. However, IoT and IIoT both have an interesting benefit, which is the fact that the relationships between the actors are weak. In the IoT platform, signals are independent. Data sharding can be used for storage purposes, and parallel computation can be used to improve computational performance. On a typical ERP or e-commerce application, data should remain centralized and connected. The following diagram highlights these differences:


ERP or e-Commerce Data Model

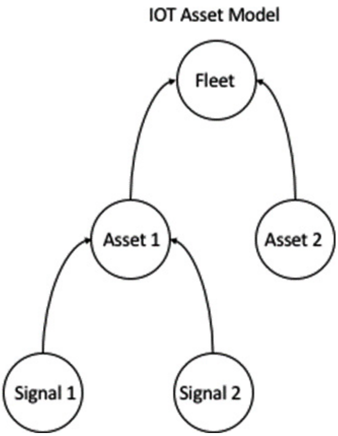

IOT Asset Model

Figure 7.1 – The data structure

This feature of the IoT platform makes it more like a NoSQL database than a SQL database, more like a microservice-based application than a monolithic application, and more like a cloud- or fog-based system than a centralized system. Therefore, IoT is more related to an ecosystem of services and applications than to an actual application.

Now that we know how the IIOT platform and its architectures work, let's look at some other features. We will cover microservices, containers, and serverless computing.

Microservice, containers, and serverless computing

When the IIoT application was being developed in 2010, the decision was made to use Apache Karaf, Java, RabbitMQ, Redis, and a historical NoSQL database (https://ieeexplore.ieee.org/abstract/

document/5598097). Apache Karaf is an OSGi application server. OSGi Alliance was founded in 1999 but only became popular in 2008. The OSGi specification, which is currently version 8.0, describes a service platform and modular system for the Java programming language that implements a complete and dynamic component model. In an OSGi framework, the same bundle (the deployment component of an application) can coexist with different versions. A strong security mechanism allows you to define the boundaries of an application, thereby avoiding unwanted spaghetti architecture. These concepts were adopted from Java 9.

OSGi was chosen due to its high degree of modularity. However, after two years of development and integration with third-party applications and different languages (such as Java, Python, C/C++, C#, and Golang), it became apparent that the spaghetti architecture was still present. IIoT is not a simple application but an ecosystem of services and computational capabilities. After a period of consideration, the IoT community decided to adopt a microservices-based architecture. The question remains, therefore, what is a microservice ?

Microservices-based architecture is nothing new. It refers to the application of high modularity to a service-oriented architecture (SOA). The idea behind microservices is quite simple – multiple, autonomous, small services communicating with each other through a web service – typically, a REST API. This is demonstrated in the following diagram:

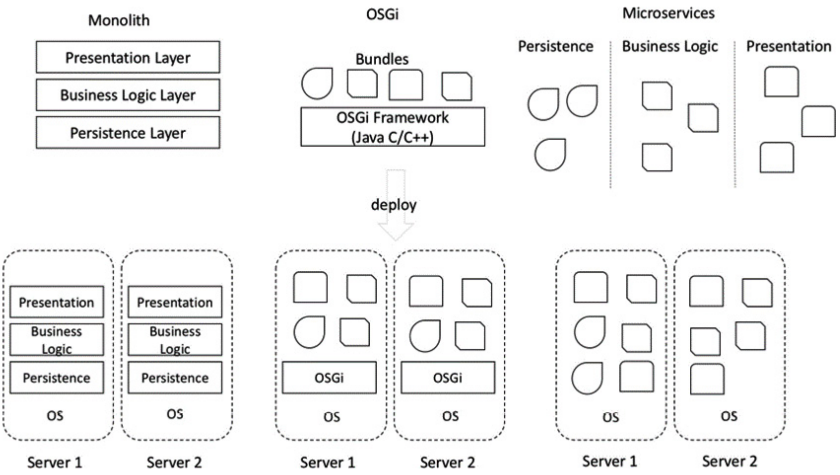

Monolith
Presentation Layer
Business Logic Layer
Persistence Layer
OSGi
Bundles
OSGi Framework (Java C/C++)
deploy
Microservices
Persistence
Business Logic
Presentation
Presentation
Business Logic
Persistence
OS
Server 1
Server 2
OSGi
OS
Server 1
Server 2
OS
Server 1
Server 2

Figure 7.2 – The journey from monolith architecture to microservices

The concept of microservices was introduced by Netflix and later formalized by Newman and Fowler (https://martinfowler.com/articles/microservices.html). Their key benefits are as follows:

- Small : Microservices are small enough to be compliant with the Single Responsibility Principle (SRP), which is one of the SOLID design principles.
- Autonomous : Microservices can be deployed as a singleton on a single machine.
- Resilient : Microservices are resilient to failure.
- Scalable : Microservices are typically deployed on a Platform-as-a-Service (PaaS) architecture. Here, an orchestrator instantiates multiple instances of the same microservice.
- Heterogenous : Microservices can be developed with any technology (including Java, Golang, and Python).
- Easy to deploy : Microservices can be deployed with a single action.
- Aligned to an organization : Different teams can develop their own microservice and share only the interfaces with other teams.
- Composable : Microservices can be consumed in different ways for different purposes.

- Replaceable : Microservices are easy to replace with different implementations.

Microservices also offer another important feature, which is the flexibility to develop evolutionary architectures.

Obviously, there are a few pitfalls. Microservices-based architecture requires a complex infrastructure. We need an orchestrator to allocate resources to services, monitor the health of the deployed components, and perform countermeasures. Microservices communicate through networking (a RESTful API or stream socket), while monolithic applications use memory.

In an IIoT infrastructure, we need scalability and high network utilization. An infrastructure oriented toward small and replaceable services, therefore, is the right choice. Another important issue is that computational capability is asset-centric. We rarely need to perform aggregation of data between assets, but it is common to measure a single asset (such as a train, car, turbine, or subsystem) rather than a single measure. This feature makes horizontal scalability easy to implement.

If we decide to adopt microservices, what happens to the old legacy applications developed in previous years? How do we convert them to microservices? Refactoring an entire application would require a huge amount of effort, which would not be approved by a company. To solve this problem, a new technology called containers was developed.

Containers allow you to package an old application as a single autonomous service – not quite a microservice but a good starting point. The most common way of implementing this is by using Docker.

Docker

Docker is a container-based platform that can host one or more components (such as databases, services, or queues) in a single bundle. Docker is like a virtual machine (VM) but shares the operating system and libraries with the host machine, instead of emulating hardware. For this reason, Docker is more like a dedicated environment than a VM. Other container-based technologies are available on the market, such as Podman, rkt, LXD, LXC, and the Open Container Initiative (OCI), and it is likely that over the next few years, we will see these technologies develop massively. For now, however, Docker is the standard technology used to develop microservices using container-based architecture.

Docker offers the great advantage of allowing us to build a container from a legacy application, so we can port old applications into a new microservice-based ecosystem with very few changes. Docker has become so popular in the past few years that, today, it is not only used to port old technologies but also to build new ones. For instance, some authors use

Docker to build analytics.

Let's consider an example. Imagine you have an old legacy application developed in FORTRAN with a reading file for input, a file database for parameters, and a writing file for output. You want to deploy this application as a scalable service. You can build your Docker container by packaging all the different elements and exposing it as a service in your infrastructure.

Let's build our first microservice with Docker. The purpose of this exercise is to familiarize yourself with Docker and Dockerfiles; we will use Docker in the next chapters extensively.

1. Install Docker Desktop Edition from https://www.docker.com/products/docker-desktop/ .
2. Then, create a local directory.
3. Create a file called `Dockerfile`.
4. Finally, copy into the file the following command-line instructions. Note that the following code runs a local Python HTTP server on your machine:

```
FROM alpine
# Install python3
RUN apk add --update python3
# Copy html
RUN mkdir /src
ADD static/ /src
# Run http server on port 8080
EXPOSE 8080
CMD ["python3", "-m", "http.server",
"8080"]
```

From the command console, you have to build a simple HTML page called **`index.html`** :

```
$ mkdir static
$ echo "<html><body>IIoT</body></html>"
> static/index.html
```

Now, follow these steps:

1. Build your Docker container with the following command:

   ```
   $ docker build --tag micropy .
   ```

2. Run your Docker container:

   ```
   $ docker run -p 80:8080 micropy
   ```

3. Open your browser to http://localhost/src or **`http://<your ip address>/src`** .

We have now built our first Docker microservice. On the Dockerfile, we provided the following instructions to Docker:

1. Use the alpine image and install Python 3
2. Create a local `src` directory
3. Copy on the local directory the content you can find in the static local folder
4. Run the Python server

We then run the command to build the container and start mapping from port `8080` to `80`. The most important thing is being able to build a container from a simple script written on the Dockerfile.

Here are some more commands and their functions:

`docker run <image>`	Runs the container
`docker ps`	Lists the containers
`docker stop <id>`	Stops the containers
`docker rm <id>`	Removes the container
`docker exec ...`	Attaches to the container
`docker build .`	Builds the container

docker compose	Starts a group of containers
docker system prune	Clean unused Docker images, networks, and storage

Table 7.1 – Common Docker commands

Docker Compose

> *Docker supports a tool called* ***Docker Compose*** *. Docker Compose allows you to build a group of Docker containers where the definition has been stored on a YML definition file. See* *https://docs.docker.com/compose/* *.*

Scaling up

> *These technologies need an orchestrator to allocate resources and manage scalability. Some of the most common solutions are Kubernetes, Swarm, EC2, and ECS. When we talk about containers, we refer to* ***Containers as a Service*** *(* ***CaaS*** *). We will explore these technologies later in Chapters 8 , 9 , and 10 .*

Microservice-oriented technologies

If you develop your own microservice, the following table will be useful, as it provides a list of the most common technologies used to build microservices:

Technology	Purpose
Spring Boot	Serving
Node.js	UI
NGINX	UI and balancing
Python	Analytic and service
Golang	Serving
.NET	UI and service
Docker	Container

Table 7.2 – Microservice-oriented technologies

In a typical software system, we must consider five important steps – development, deployment, operation, maintenance, and enhancement. The final two, maintenance and enhancement, are the most difficult elements in IIoT. Developers are often asked by their business department to implement small functionalities quickly, or to build workarounds to manage a particularly difficult device or asset. Building small services and interconnecting each of them allow architects and developers to build a scalable system easy to maintain and improve. The only pattern to keep in mind is the single responsibility pattern, meaning each service must have a defined and unique purpose.

Software as a service

Cloud providers (such as AWS and Oracle) have proposed serverless technologies. The basic idea of these is to provide a developer with the ability to build and deploy a function. In other words, the developer can build their own function and deploy it in a single click, without having to redeploy the entire microservice. Serverless technologies can be considered an extreme version of microservices. When we discuss functions, we refer to Functions as a Service (FaaS). See the following table:

Function	
Infrastructure as a Service (IaaS)	

Cloud orchestrators	
Provides access to a resource (CPU, memory,	

	or OS) in a virtualized environment.
PaaS	Provides an environment to build applications and services for the developer.
Software as a Service (SaaS)	Provides access to services and applications ready to be used.
Backend as a Service (BaaS)	A computing model that serves as the middleware that allows developers to connect applications to cloud services through the API or SDK.
Data as a Service (DaaS)	An extension of SaaS where content and data, such as wheat- ear, city traffic, or statistics, are provided as a service.
CaaS	An extension of PaaS where the developer can deploy container-based applications.
FaaS	An extension of PaaS where the developer can build a small function without worrying about the application server or the OS.
X as a Service (XaaS)	Anything as a service. It is a way to generalize the concept of *as a service* .

Table 7.3 – The different types of cloud

orchestrators

These technologies are not strictly related to IoT or IIoT, but they provide a good point of reference for implementing an IIoT application. We will explore serverless technologies, such as AWS or Azure, in *Chapters 10* and *11* . Now, it's time to understand the IIoT data flow in detail.

The standard IIoT flow

In *Chapter 2* , we looked at how industrial data is generated, gathered, and transferred to the cloud by familiarizing ourselves with computer-integrated manufacturing (CIM). We also looked at industrial equipment, such as DCS, PLCs, and SCADA, and network protocols, such as Fieldbus, ControlBus, and OPC. This gave us a picture of the whole path of industrial signals, from when they are generated by sensors in a factory to when they are processed in the cloud. In *Chapters 3* and *4* , we talked in detail about data acquisition, data sources, related protocols, and the edge devices used to push data into the cloud. This has shown us how data acquisition is the foundation of the IIoT and how we need to be compliant with several security and safety standards. Data, therefore, must be analyzed and processed. In a typical new IIoT architecture, there are other parallel mechanisms, such as central data storage and data processing and execution.

The following diagram shows the flow of IIoT data on the cloud side, summing up what was already presented in *Chapter 2* :

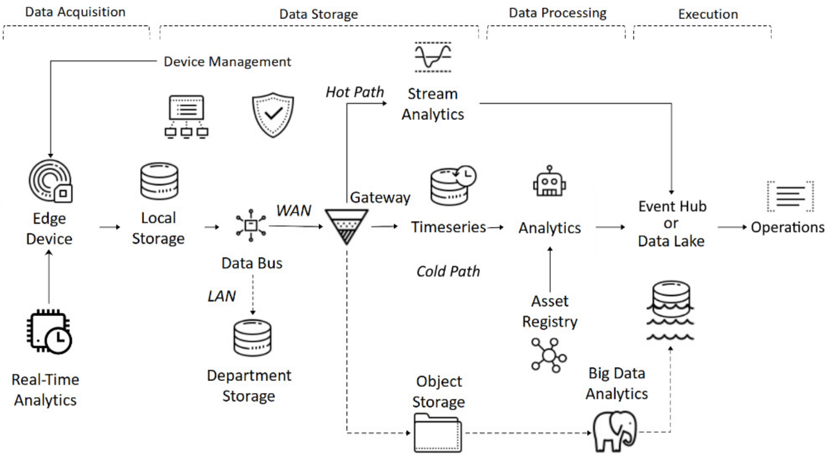

Data Acquisition
Data Storage
Data Processing
Execution
Device Management
Hot Path
Stream Analytics
Edge Device
Local Storage
WAN
Gateway
Data Bus
Timeseries
Analytics
Event Hub or Data Lake
Operations
Cold Path
LAN
Asset Registry
Real-Time Analytics
Department Storage
Object Storage
Big Data Analytics

Figure 7.3 – End-to-end data flow on the IIoT

During data transfer, the data coming from the sensors is gathered from the data sources (such as PLCs, DCSs, SCADA, or a historian) and stored temporarily to avoid data loss due to a connectivity issue. The data can be time-series data, such as an event, semi-structured data, such as a log or binary file, or completely unstructured data, such as an image. Time-series data and events are collected frequently (from every second to every few minutes). Files are normally collected when they are triggered locally, which can happen through machine shutdown or inspection. Shadow data can then be sent, through LAN, to the department data center. The data is then sent by the edge over the WAN and stored in a centralized data lake and a time-series database (TSDB). The data lake can be cloud-based, an on-premises data center, or a third-party storage system.

Data can be immediately processed using data-stream analytics, which is called a hot path , with a simple rule engine platform based on the threshold or the smart threshold.

Advanced analytics, including digital twins, machine learning, deep learning, and data-driven or physics-based analytics, can process a large amount of data (from ten minutes to one month's worth of data) coming from different sensors. This data is stored on an intermediate repository (called a cold path , as can be seen in the preceding diagram). These analytics are triggered by a scheduler or the availability of the data, and they need a lot of computational resources and dedicated hardware, such as a CPU and GPU.

> ### *Cold paths and hot paths*

> *Azure refers to cold paths and hot paths. In a hot path, data is processed immediately. In a cold path, data is stored and processed later. Normally, we use data streams for hot paths and micro-batch analytics for cold paths.*

These analytics often need additional information, such as the model of the machine being monitored and the operational attributes; this information is contained in the asset registry. The asset registry has information about the type of asset you are monitoring, including its name, serial number, symbolic name, location, operating capabilities, the history of the parts it is made up of, and the role it plays in the production process. In the asset registry, we can store the list of measures of each asset, the logic name, the unit of measure, and the boundary range. In the industrial sector, this static information is important so that the correct analytic model can be applied.

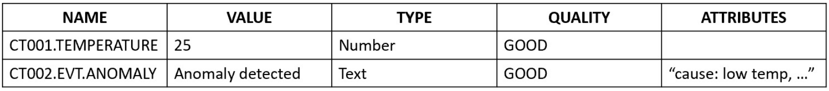
The output of the analytics, either stream-based or advanced, is typically calculated data. This might be the expected performance, the operability drift, an alarm such as an anomaly vibration, a recommendation, such as clean, replace filter, or close valve, or a report. These insights can be analyzed by an operator to provide information to the operations manager. They might then decide to open a work order to improve the process. Although these operations are not part of an IIoT process, the analytics should skip this data because the signals are unpacked, and therefore, nothing useful can be concluded. For example, if you decide to turn off a machine whose status is being monitored, this should be interpreted as an anomaly and the analytics should raise an alert that something is wrong.

Occasionally, IIoT can use *big data analytics* to process a huge amount of data, such as images or raw data. These technologies can employ data lakes and big data platforms.

In the next section, we will explore all the most important components of the data flow described here.

Understanding the time-series technologies

The IIoT process requires a continuously high data flow from the machines to the central system. This cannot be supported by a standard SQL database. For this reason, with IIoT, we prefer to use technologies inherited from big data, such as NoSQL databases. NoSQL databases are storage systems that can manage big data very efficiently by providing low support for aggregation functions and relationships among data. In the IIoT, the most common database is the historical database, which can store events, such as alerts or signals and measures, using a time-based method, where the timestamp is the key, or one of the keys, of the item. The most common historical database is the TSDB. A time series is a series of observations captured over a period of time, but in the IIoT, it has to support more information. A typical time-series-based item would look as follows:



NAME	VALUE	TYPE	QUALITY	ATTRIBUTES
CT001.TEMPERATURE	25	Number	GOOD	
CT002.EVT.ANOMALY	Anomaly detected	Text	GOOD	"cause: low temp, …"

Figure 7.4 – A time-series item example

The **`NAME`** column refers to the logic name of the item, the **`VALUE`** field stores the measured value, and the **`QUALITY`** column, which can be **`GOOD`** , **`BAD`** , **`UNKNOWN`** , or **`ERROR`** , stores information about the data acquisition. We can sometimes store additional information, such as the attribute and data type.

The following is a list of the most common TSDBs:

- OSIsoft PI : This is a native, commercial TSDB database developed by OSIsoft. It is very efficient and provides support for data collecting and data streaming. This technology is often used for local storage, departmental storage, and remote storage in the industrial sector. OSI PI uses a proprietary protocol to send data between departments. One of the most interesting features of OSI PI is its ability to support analytics and to make historical and real-time data instantly accessible to users.
- Uniformance Process History Database (PHD) : Uniformance PHD is a Honeywell product that has high integration with Distributed Control Systems (DCSs).
- OpenTSDB : OpenTSDB is based instead on HBase. OpenTSDB uses the RESTful API and Telnet to query and ingest data.
- Riak TS (RTS) : This is a NoSQL TSDB optimized for time-series data.

It ingests, transforms, and stores massive amounts of time-series data with good scalability. It is integrated with the most popular languages and technologies.

- Prometheus : Prometheus is a database used to store time-series data for KPI. It is distributed under the Apache License.
- InfluxDB : InfluxDB is a very fast time-series database distributed under an open source license with commercial support. InfluxDB allows precision to a nanosecond. We will see how to connect IndfluxDB, NODE-Red, and a simple OPC-UA server at the end of this chapter.
- Timescale : This is an extension of PostgreSQL database to empower this popular database with support for time series.
- Elasticsearch : This is a popular, document-based database for fast indexing and searching. It is not a real TSDB, but thanks to its high indexing performance and its ability to integrate with Kibana, it can be used to visualize data.
- Cloud-based TSDBs : In the next chapters, we will look at some of the solutions proposed by AWS and Azure for TSDB. Generally, these solutions support TSDBs based on NoSQL and SQL databases and extend the schema with the API.

Not only sensors

Numerical sensors and alerts are the most common data of industry; however, it is not so unusual to send images, vibrations, videos, and text. Time-series databases can support also binary or composite datatypes such as JSON.

A TSDB is the most important component of IIoT. In the next section, we will study another very important component, the asset registry.

Asset registry

Asset registry is the second most important repository of the IIoT. On the asset registry, we need to store information about our assets and the relationships between them. This information is normally imported from enterprise resource planning (ERP) or a computerized maintenance management system (CMMS). Assets are linked together by membership or production-flow relationships.

A very simple example of an asset structure is shown in the following diagram:

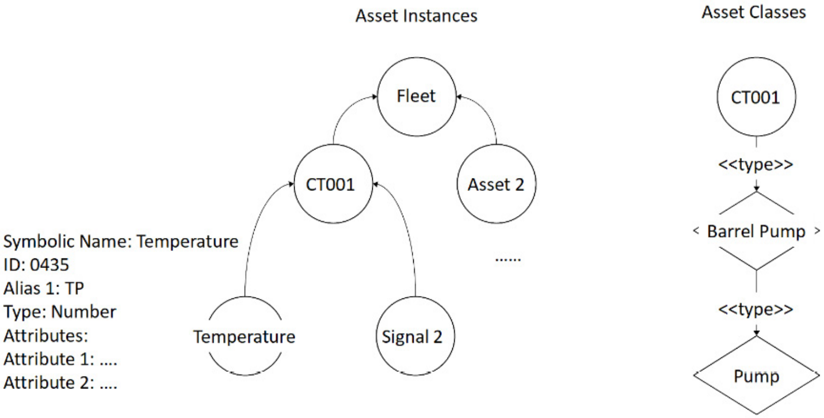

Asset Instances
Asset Classes
Fleet
CT001
Asset 2
Symbolic Name: Temperature
ID: 0435
Alias 1: TP
Type: Number
Attributes:
Attribute 1: ....
Attribute 2: ....
......
Temperature
Signal 2
CT001
<<type>>
Barrel Pump
<<type>>
Pump

Figure 7.5 – An asset hierarchy example

The asset structure is made up of the following:

- Asset instances : The CT001 asset is a barrel pump that is part of the fleet, which may include a subsystem, plant, or factory. We collect several measures from it, such as temperature. Every node of the asset should expose some simple or structured properties, such as asset type, aliases, name, ID, and custom attributes.
- Asset classes : Through classification, you can define the type of asset using an object-oriented methodology.

The asset registry is normally a graph or a relational database where information about the asset and measures is stored to be monitored and collected. The following is a list of technologies that can be used to implement our asset registry:

- Neo4j : A NoSQL database with native support for graphs. Neo4j can be used to model the asset registry and the relationship between assets.

- MongoDB : A document-based (JSON) database. We can use it to store information about our asset list and measures.
- Standard SQL : This can be used to model our asset hierarchy using the relationship between tables, but it cannot be used to define a very flexible structure.

We have covered the asset registry and TSDB components. Let's see in the next section how to extract insights from the data.

Data processing and the analytics platform

Analytics are the active component of the IIoT flow able to generate value. In the IIoT, analytics are classified as descriptive, diagnostic, predictive, or prescriptive. Let's learn more about each of these:

- Descriptive analytics analyze live data to provide a comprehensive snapshot of the current status.
- Diagnostic analytics provide information about an event due to an anomaly.

- Predictive analytics add the word *when* to diagnostic analytics and try to predict future events. Predictive analytics not only provide information about the event that is going to happen but also when it will happen, the risk of impact, and the probability that it will occur.
- Prescriptive analytics provide insights about the action that should be taken.

There is some overlap between these classes. They are summarized in the following table:

[illegible] use cases
[illegible] happen?
[illegible] The probability that the event will occur.
[illegible] and reinforcement learning.

Table 7.4 – Analytics types

Generally speaking, prescriptive analytics are the most complex analytics and descriptive analytics are the simplest.

Due to the large amount of data and the correlation between measures, we need at least two levels of data-processing analytics – excursion monitoring analytics (EMAs) and advanced analytics.

Excursion monitoring analytics

Excursion monitoring analytics (EMAs) are the first layer of analytics, which are normally used for descriptive, diagnostic, or short-term predictions. Signals are acquired and sent to a rule engine directly using a data-stream mechanism. Simple rules analyze signal by signal to discover any potential deviations from standard behavior. A simple threshold rule is a common example. EMAs are not as simple as they might appear – the engine should avoid raising the same event multiple times within a time range, due to oscillations of the signal. The following diagram shows a typical example:

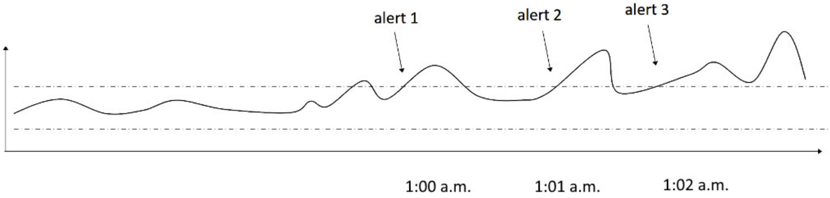


Figure 7.6 – An example of an EMA raising the

same event multiple times in a few minutes

EMAs can also monitor calculated measures that are made up of raw measures. In an industrial context, we refer to raw measures to describe the measures acquired from a sensor before conversion. More sophisticated rule engines tune the threshold according to data acquired in the past, or from user feedback.

From a technological point of view, there are sophisticated commercial products, such as OSI PI, InfluxDB alerts, and Azure PI Integrator, that cover these use cases. These solutions can be installed on the edge or in the cloud.

The fastest way to develop our rule engine solution is to use a serverless solution. We will see an example of this in the next chapters.

Advanced analytics

The second level of analytics is advanced analytics, which is used for prediction and prognosis. An example of this type of analytics is a digital twin , which tries to replicate the physical model of an asset through a mathematical formula or a data-driven model. A Digital Twin is a digital representation of an asset or system across its life cycle, which can be used for various purposes.

Advanced analytics requires a more complex platform. It processes a relatively large amount of data (such as 30 to 60 measures over the last 20 minutes to one month) and compares it with a digital model or the expected behavior. Advanced analytics requires more information, such as a model of the asset, its operability, the digital model (which is learned or defined previously), and the status of the previous analysis. Common technologies used to develop these analytics are R, Python, Giulia, Scala, and MATLAB. Engines hosting these analytics are cloud-based platforms or big data platforms.

> ***The language for analytics***
>
> *Today, thanks to the AI explosion (TensorFlow and PyTorch), Python is the de facto language for analytics. C/C++ can be used for fast edge analytics. MATLAB is a good tool thanks to its edge interoperability.*

Big data analytics

We do not normally need big data analytics in IIoT, but we might occasionally need to process a large amount of data, such as images, log files, or binary files, for security purposes or production and supply-chain optimization. If you'd like to learn more about big data analytics, you can do so by visiting https://www.ibm.com/topics/big-data-analytics .

EMAs are good candidates for hot paths, whereas advanced analytics and big data analytics are usually used for cold paths.

In the next section, let's experiment with InfluxDB and Node-RED.

Hands-on with InfluxDB, Node-RED, and OPC UA

Now, let's implement a flow that reads the CPU used by the OPC UA server implemented in *Chapter 6* . We will use Node-RED as the edge, write the signal to InfluxDB, and display it with the InfluxDB UI. We will leverage Docker to build the infrastructure.

The first step is to clone the official repo on a local PC:

```
$ git clone https://github.com/
PacktPublishing/Hands-On- Industrial-
Internet-of-Things-Second-Edition
```

Then, change the current directory to the `Chapter 7` directory:

```
$ cd Hands-On-Industrial-Internet-of-
Things-Second-Edition/ Chapter7
```

Once you've cloned the repo, follow the following steps:

1. Open command prompt and start the `opc-ua` server:

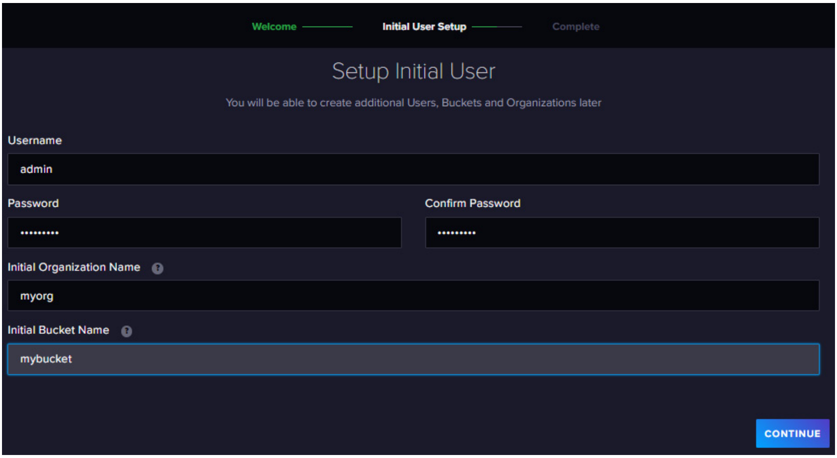
   ```
   $ cd node-opcua-sampleserver
   $ npm install
   $ npm start
   ```

   Note that it is the same procedure implemented in *Chapter 6* .

2. Start the InfluxDB server using `docker` :

   ```
   $ mkdir data
   $ docker run -p 8086:8086 -v data:/
   var/lib/influxdb2 --name=mytsdb
   influxdb:latest
   ```

3. Open the browser to `http://localhost:8086` . When we start for the first time, we have to define the username, password, organization, and bucket name :



Setup Initial User

You will be able to create additional Users, Buckets and Organizations later

Username: admin

Password: •••••••••

Confirm Password: •••••••••

Initial Organization Name: myorg

Initial Bucket Name: mybucket

Figure 7.7 – The first configuration

During the second screen, InfluxDB will provide a token. Remember to save the token, as we will reuse it later:

1. Start the Node-RED server using `docker`.
2. From the command line, enter the following command:

```
$ mkdir data
$ docker run -it -p 1880:1880 -v
data:/data --name mynodered nodered/
node-red
```

3. On a separate command line, run the following:

```
$ docker exec -it mynodered npm
install node-red-contrib-opcua
$ docker exec -it mynodered npm
```

```
install node-red-contrib-influxdb
$ docker commit mynodered iiot-
book-2/node-red-opc-influxdb
```

4. Close the running docker and start a new Docker container with the new image:

```
$ docker run -it -p 1880:1880 -v
data:/data --name mynodered iiot-
book-2/node-red-opc-influxdb
```

Note that it is the same procedure implemented in *Chapter 6* ; we simply added the InfluxDB plugin.

> ### *docker commit*

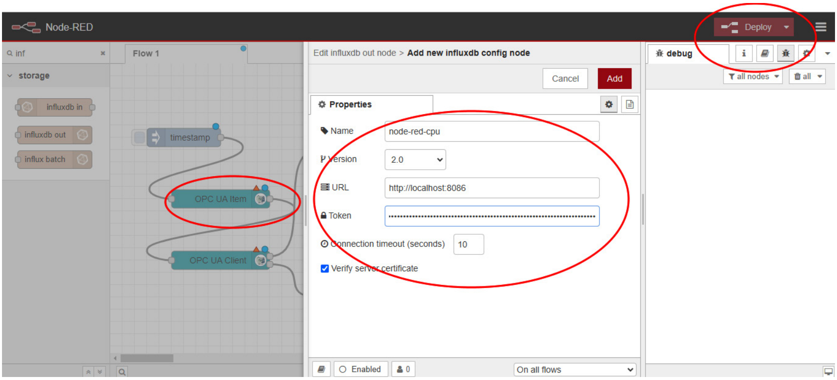
> `docker commit` *allows us to save the changes made to the running container on a new image, ensuring that we will not lose them if we restart the container. Is this the correct way? No, it is better to build a Dockerfile.*

Now, you can connect to Node-RED on http://localhost:1880/ and see the OPC UA and InfluxDB plugins installed.

Now, we can figure out the destination of the `opc-`

`ua` client node of Node-RED. Follow these steps:

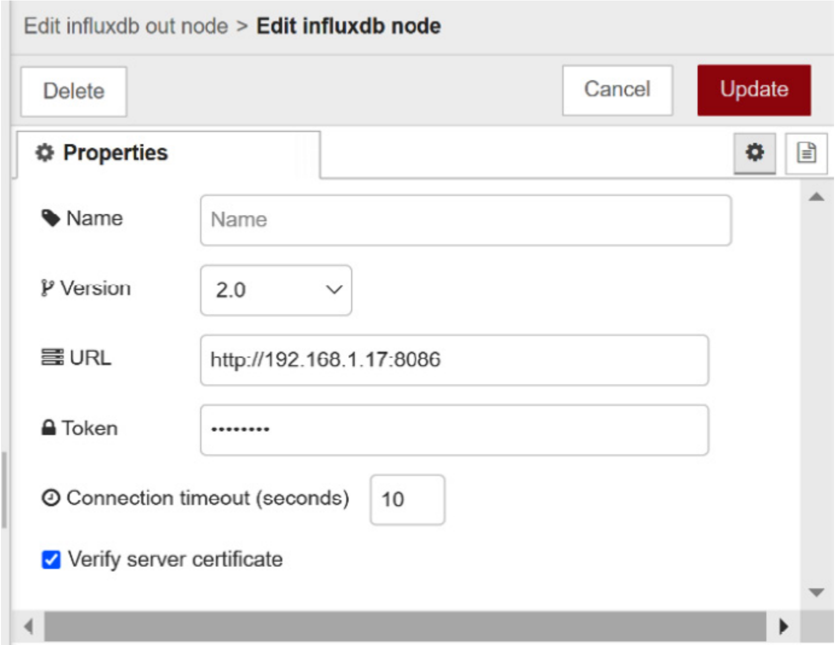
1. From the left command panel, search for `InfluxDB`, and drag and drop the `InfluxDB` out node onto the current flow, as depicted in the following figure:


Node-RED
Deploy
inf
storage
influxdb in
influxdb out
influx batch
Flow 1
timestamp
OPC UA Item
OPC UA Client
Edit influxdb out node > Add new influxdb config node
Cancel
Add
Properties
debug
all nodes
all
Enabled
0
On all flows

Name: node-red-cpu
Version: 2.0
URL: http://localhost:8086
Token: ••
Connection timeout (seconds): 10
☑ Verify server certificate

Figure 7.8 – Configure the InfluxDB destination

1. Double-click on the node, click on the *pencil*, and then configure a version of InfluxDB and server IP (use the host server IP).
2. In the Token field, copy and paste the token saved during the InfluxDB startup:


Edit influxdb out node > **Edit influxdb node**

Name: Name

Version: 2.0

URL: http://192.168.1.17:8086

Token: •••••••••

Connection timeout (seconds): 10

☑ Verify server certificate

Figure 7.9 – Configure the influxdb server

1. Save and define the measurement name, `node-red-cpu`.
2. Now, open the InfluxDB Data Explorer at `http://localhost:8086;` on the bottom bar, search for mybucket | node-red-cpu | value | SUBMIT .

The following figure shows the expected result:

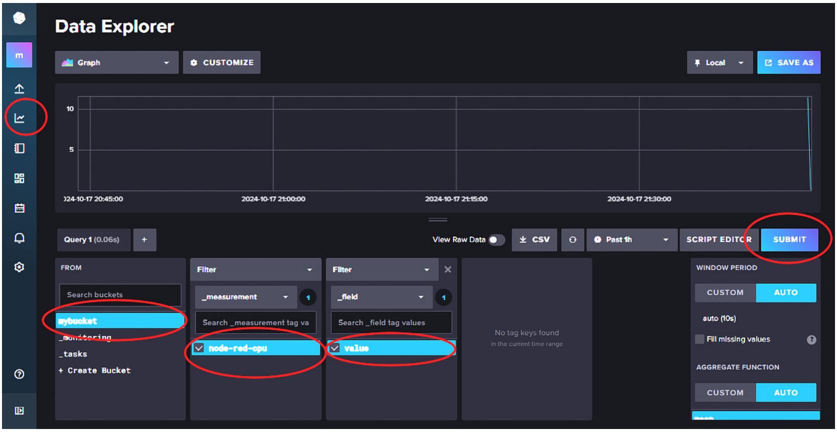

Data Explorer
Graph
CUSTOMIZE
Local
SAVE AS
10
5
)24-10-17 20:45:00
2024-10-17 21:00:00
2024-10-17 21:15:00
2024-10-17 21:30:00
Query 1 (0.06s)
+
View Raw Data
CSV
Past 1h
SCRIPT EDITOR
SUBMIT
FROM
Search buckets
mybucket
_monitoring
_tasks
+ Create Bucket
Filter
_measurement
Search _measurement tag va
node-red-cpu
Filter
_field
Search _field tag values
value
No tag keys found
in the current time range
WINDOW PERIOD
CUSTOM
AUTO
auto (10s)
Fill missing values
AGGREGATE FUNCTION
CUSTOM
AUTO

Figure 7.10 – The InfluxDB UI console showing the CPU utilization of the OPC UA node

Well done! You have built your first minimal local IoT platform! But what exactly did we do? Let's recap:

1. During the first step, we started the `opc-ua` node server
2. Then, we configured NodeRED with two plugins – `opc-ua` and InfluxDB
3. We reused the flow from *Chapter 6* and connected the output of NodeRED to InfluxDB

In other words, we have built the following architecture:

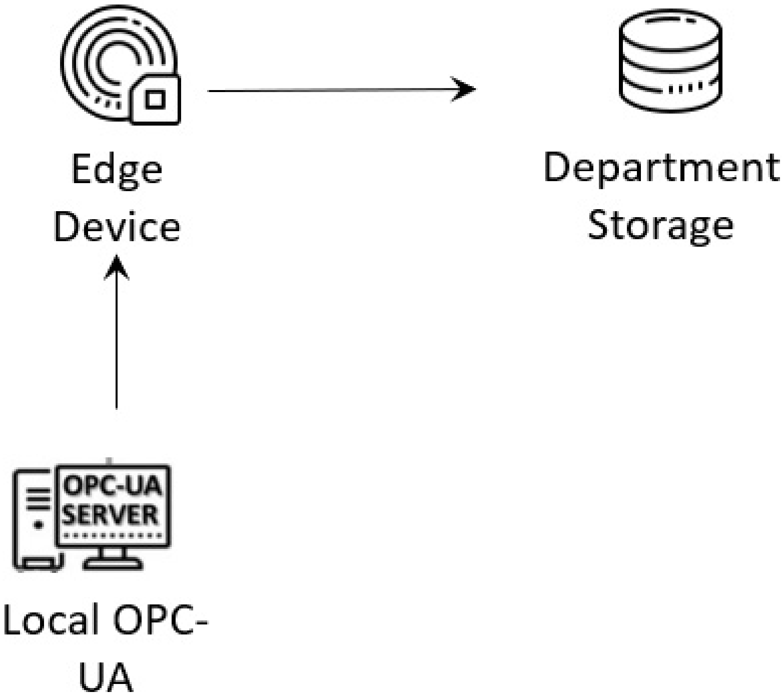

Edge Device
Department Storage
OPC-UA SERVER
Local OPC-UA

Figure 7.11 – A simple IoT local flow

> ***Tip***
>
> *If you get the* ***Error response from daemon: Conflict. The container name "YOUR CONTAINERNAME" is already in use by container...*** *error, delete it with the* `docker rm YOUR CONTAINERNAME` *command.*

Using Docker Compose and

Docker Build

Now, let’s do the same exercise again but use Docker Build and Docker Compose . It is important to also do this exercise to speed up deployment in the following chapters:

1. From the command console, run the following command:

```
$ cd Hands-On-Industrial-Internet-
of-Things-Second-Edition/ Chapter7
```

2. Then, build the two Dockerfile provided with the following:

```
$ docker compose build
```

3. Then, we can start the three servers using the following:

```
$ docker compose up
```

The Dockerfile to build the OPC UA server is the following:

```
FROM node:alpine
RUN apk add --update openssl
RUN mkdir /app
WORKDIR /app
COPY package.json /app
```

```
RUN npm install
COPY . /app
EXPOSE 26543
CMD ["npm", "start"]
```

The first line inherits the current image from the official image of Node.js (https://hub.docker.com/_/node/); if the image is not available on the local PC, Docker will download it from Docker Hub.

In the second line, we install `openssl`. From lines 4 to 6, we install the Node.js packages and the current application. The last line starts the Node.js server.

The Dockerfile to build the `node-red` edge server is the following:

```
FROM nodered/node-red
RUN npm install node-red-contrib-opcua
RUN npm install node-red-contrib-
influxdb
EXPOSE 1880
CMD ["npm", "start"]
```

The first line inherits the current image from the official image of `node-red` (https://

hub.docker.com/r/nodered/node-red/) ; if the image is not available on the local PC, Docker will download it from Docker Hub.

From lines 2 to 3, we install the Node.js packages. The last line starts the **`node-red`** server.

`docker-compose` , reported in the following code block, instructs Docker to start three services and map each server to a specific port:

```
version: '2'
services:
  nodered:
    image: iiot-book-2/node-red-opc-
influxdb
    build: ./node-red
    ports:
        - "1880:1880"
    volumes:
        - ./data:/data
  influxdb:
    image: influxdb:latest
    ports:
      - '8086:8086'
    volumes:
      - ./data:/var/lib/influxdb
    environment:
      - INFLUXDB_DB=db0
      - INFLUXDB_ADMIN_USER=amin
      -
INFLUXDB_ADMIN_PASSWORD=xxxxxxxx
  opc-ua-server:
      build: ./node-opcua-sampleserver
      ports:
        - "26543:26543"
```

The big advantage of Docker Compose is being able to build a single ecosystem in a single shot; indeed, Docker Compose creates a local network so that every server can be connected using a domain-specific name. The following diagram shows the new flow, where we replace the local IP with the fully qualified name of the servers:

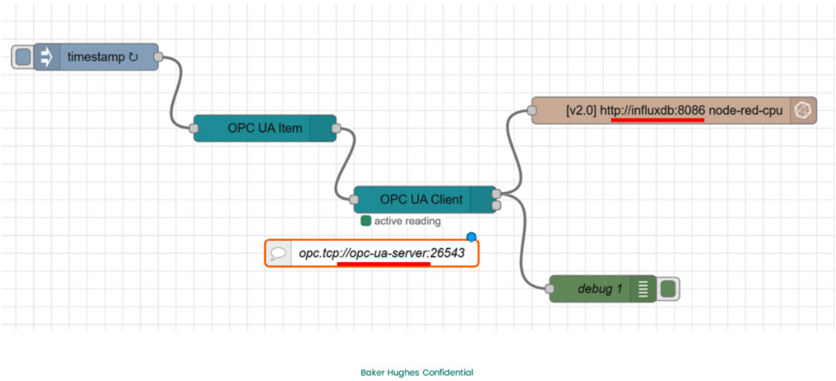

timestamp ↻
OPC UA Item
[v2.0] http://influxdb:8086 node-red-cpu
OPC UA Client
active reading
opc.tcp://opc-ua-server:26543
debug 1
Baker Hughes Confidential

Figure 7.12 – The Node-RED data flow

We have just built our first data flow; even if it cannot be defined as a canonical IoT flow, it is the seed on which we will build a more complex architecture.

Summary

In this chapter, we looked at the basic concepts of IIoT data processing. First, we explored the concept of microservices versus a monolithic application. We studied Docker and how it can help us deploy microservices-based architectures. In the upcoming chapters, we will make extensive use of Docker to simplify deployment.

We learned about the key principles of data storage, including time series, asset management using asset registry, and data processing with advanced analytics. Finally, we implemented our first end-to-end IoT flow with InfluxDB, Node-RED, and OPC-UA.

In the next chapter, we will build a real IIoT solution based on common open source technologies.

8

Implementing a Custom Industrial IoT Platform

In *Chapter 7* , we described the general flow of processing IIoT data. In this chapter, we are going to update our flow, using the most common open source technologies to develop our platform from scratch. These include the following:

- InfluxDb as a time-series database (TSDB)
- Neo4j as an asset database
- Mosquitto to collect data and adapt
- Airflow as an analytics platform
- Grafana to visualize data

The purpose of our exercise is not to develop a real I-IoT platform but, instead, to discuss some key topics related to IIoT, and to introduce various technologies that we can use in our on-premises IIoT stack. At the end of this chapter, we will compare this solution with other open source solutions, such as Kaa IoT and Eclipse IoT.

In this chapter, we will cover the following topics:

- Developing a platform based on open source technologies
- Using Mosquitto as an MQTT connector
- Using Airflow as a data dispatcher
- Storing time-series data on InfluxDB
- Building an asset registry to store asset information
- Understanding the pros and cons of the proposed platform

Technical requirements

In this chapter, we need the following:

- Docker Desktop Edition: https://www.docker.com/
- Git: https://git-scm.com/downloads
- Node.js: https://nodejs.org
- Python3: https://www.python.org/downloads/
- Visual Studio Code (optional): https://code.visualstudio.com/

To get a complete solution for the exercises, take a look at the repository at https://github.com/PacktPublishing/Hands-On-Industrial-Internet-of-Things-Second-Edition :

```
$ git clone https://github.com/
PacktPublishing/Hands-On- Industrial-
Internet-of-Things-Second-Edition
```

An open source platform in practice

According to surveys carried out by the Eclipse Foundation, 20% of IoT platforms have been developed as on-premises solutions (https://newsroom.eclipse.org/news/announcements/eclipse-foundation-releases-2023-iot-edge-developer-survey-results). This shows a positive trend about the use of the public cloud, but there are dozens of reasons why we might want to continue to develop using on-premises or open source solutions:

- Return on investment : We might have a budget that is too small to justify a big investment
- Technology : We might want to use technology that does not depend strictly on a supplier
- Privacy : We might want to export data outside our country
- Data shadowing : We might need a copy of the data on our legacy platform, either to experiment with or just as a backup
- Integration : We might be implementing an inter-cloud platform, meaning we need an integration layer
- Experience : We might be developing our first IIoT platform, so we want to start small
- Product : We might want to develop our product using only cloud infrastructure
- Intellectual property : We might need a small space to run our analytics, helping to protect our intellectual property
- Legacy integration : We might have developed an old legacy platform that we want to integrate with our I-IoT platform
- Connectivity : We might have some restrictions on connectivity, and we might want to control the traffic

In this chapter, we will develop a custom platform by reusing the concepts that we learned in *Chapter 7* . The following figure depicts the proposed solution:

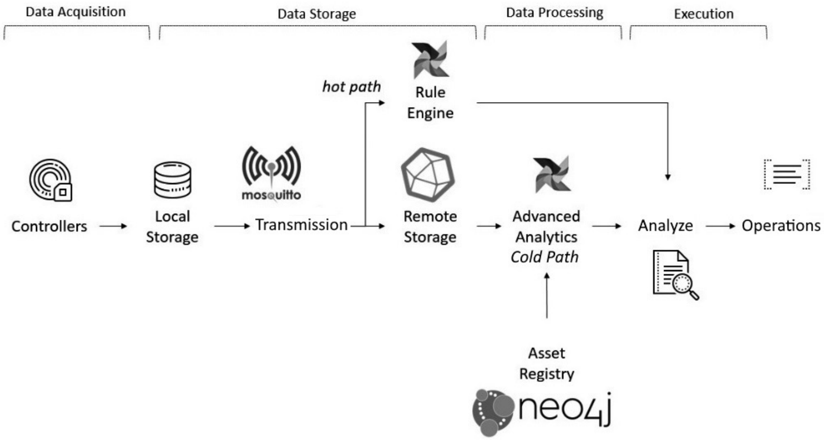

Data Acquisition
Data Storage
Data Processing
Execution
hot path
Rule Engine
mosquitto
Controllers
Local Storage
Transmission
Remote Storage
Advanced Analytics Cold Path
Analyze
Operations
Asset Registry
neo4j

Figure 8.1 – An open source IIoT architecture

We will use *Mosquitto* as a Message Queuing Telemetry Transport (MQTT) data broker in conjunction with Telegraf to improve scalability and provide an isolation layer. We can replace MQTT with different technologies, and we can also use it in conjunction with a different protocol by replacing Mosquitto only. Our data will be stored on an InfluxDB data abstraction layer. Rule-based analytics will process data through the MQTT-Airflow custom service data stream (the hot path), while advanced analytics will use Python and Airflow to process a small portion of data via micro-batch processing (the cold path). The results of these analytics can be stored on Elasticsearch or a SQL database (we won't be covering how to manage this data as part of this project – see the book *Kibana 8.x – a Quick Start Guide to Data Analysis: Learn about data exploration, visualization, and dashboard building with Kibana*). Then, we will reuse the Node-RED edge and OPC-UA simulation server to simulate data acquisition from the field. Finally, we will implement an asset registry using Neo4j.

Mosquitto as a data gateway

In the first couple of chapters of this book, we dealt with the most common protocols for data transmission. In this example, we will use MQTT. MQTT isn't the only choice for the IIoT, but it has great support for HTTPS encapsulation. In the industrial sector, we normally have many technologies, such as MQTT, Constrained

Application Protocol (CoAP), Open Platform Communications (OPC), and OPC Unified Architecture (UA), and we need to integrate these all into a single data bus. In this example, we will use Mosquitto (https://mosquitto.org/download/) for frontend data acquisition and distribution.

Let us look at installing Mosquitto as an MQTT connector. To install Mosquitto, we can follow the instructions from https://mosquitto.org/download/ , or we can use Docker as usual:

1. Configuration : Prepare the configuration.

   Execute the following from the command line:

   ```
   $ mkdir mosquitto
   $ mkdir conf
   $ mkdir data
   $ mkdir log
   ```

   Then, create a file with the name **`conf/mosquitto.conf`** with the following lines:

   ```
   persistence true
   persistence_location /mosquitto/
   data/
   ```

```
log_dest file /mosquitto/log/
mosquitto.log
log_dest stdout
listener 1883
allow_anonymous true
```

2. Start Mosquitto : From the command console, run the following command:

```
$ docker run -it -p 1883:1883 -p
9001:9001 -v mosquitto.conf:/
mosquitto/config/mosquitto.conf
eclipse-mosquitto
```

3. Start a simple client : We are going to deploy the official Mosquito distribution, so we can test our MQTT broker. We need to install an MQTT client. Run the following command with an administrative user. Execute the following from the command line:

```
$ npm install mqtt-cli -g
```

Subscribe to a topic as follows. Execute the following from the command line:

```
$ mqtt-cli localhost topic/hello
payload -w
```

Publish our first message as follows. Execute the following from the command line:

```
$ mqtt-cli localhost topic/hello
"Test hello message"
```

The outcome should look like this:

```
mqtt-cli localhost topic/hello
payload -w Topic: topic/hello
Message: payload
Topic: topic/hello Message: Test
hello message
```

4. Use Docker Compose : Instead of starting Mosquito from the command console with `docker run`, it is better to write the configuration in a `docker-compose.yml` file and use `docker compose up`:

```
mosquitto:
    container_name: mosquitto
    image: eclipse-mosquitto
    volumes:
    - "./mosquitto/config:/
mosquitto/config"
    - "./mosquitto/log:/mosquitto/
log"
    - "./mosquitto/data/:/mosquitto/
data"
    ports:
    - "1883:1883"
```

We need also to add the external network to allow interoperability among Docker Compose networks. On `docker-file compose.yml`, add the following at the end:

```
        networks:
        - iiot-book-2-net
```

To get a complete solution for the exercises, take a look at the repository at https://github.com/PacktPublishing/Hands-On-Industrial-Internet-of-Things-Second-Edition .

> ***The iiot-book-2-net network***

> *In the next sections of this chapter, we assume that a network with the name* **`iiot-book-2-net`** *has been already created in the host machine. This network has been created by* **`docker compose`**, *but if there is an error, run the following command from the command console –* **`docker network create -d bridge iiot-book-2-net`**.

Storing time-series data on InfluxDB

In the previous sections, we learned how to work with an MQTT stream. Now, we want to store our data in a scalable and fault-tolerant database. We can use various NoSQL databases, such as HBase, Parquet, Prometheus, Cassandra, and OpenTSDB, but for our platform, we are going to use InfluxDB because it supports clustering, PaaS, and business license for scaling up to the enterprise level.

Create a file with the name **`docker-compose.yml`**. Add the following lines to **`docker-compose.yml`**:

```
influxdb:
    image: influxdb:latest
    ports:
      - '8086:8086'
    volumes:
      - ./data:/var/lib/influxdb
```

Unlike the exercise in the previous chapter, here, we will configure static security on the InfluxDB start screen in the Docker Compose file. Therefore, we need to add these parameters to our **`docker-compose.yml`**:

```
environment:
      - INFLUXDB_DB=db0
      -
DOCKER_INFLUXDB_INIT_MODE=setup
      -
DOCKER_INFLUXDB_INIT_USERNAME=admin
      -
DOCKER_INFLUXDB_INIT_PASSWORD=00000000
      -
DOCKER_INFLUXDB_INIT_ADMIN_TOKEN=my-
super-secret-auth-token
      - DOCKER_INFLUXDB_INIT_ORG=myorg
      -
DOCKER_INFLUXDB_INIT_BUCKET=iiot-
book-2
```

`init_mode` , set to **`setup`** , allows us to skip the start screen when we need to configure a username and password. Also, the token is predefined so that we can reuse it on the next configurations.

We need also to add the external network to allow interoperability among Docker Compose networks:

```
    networks:
    - iiot-book-2-net
```

Graphite

> *InfluxDb supports Graphite protocol.* ***Graphite*** *is a popular time series database very common and well documented. InfluxDB supports Graphite protocol, so if you want to use Graphite protocol, visit* *https://docs.influxdata.com/influxdb/v1/supported_protocols/graphite/* .

Connecting InfluxDb and MQTT Data Broker

InfluxDB supports the Telegraf plugin; Telegraf can subscribe to MQTT Data Broker and Mosquitto and inject data faster into an InfluxDB database.

This component is much more important than it appears, as it acts as a data dispatcher within your IIoT architecture. It must support three macro-functionalities:

- Scalability : You may have to manage billions of sensors.
- Flexibility : It must be able to connect to various databases – for example, a third-party customer may require you to save data segregated from other data, or you may need to connect it to an external data broker.
- Configurability : You may have to manage a complicated naming convention method to avoid conflicts among devices.

Telegraf (https://github.com/influxdata/telegraf) can support all the previous requirements. To configure Telegraf, we can add another server to **`docker-compose.yml`** :

1. Configure Telegraf : We need to add a configuration file to map data from Mosquitto to InfluxDB.

   Execute the following from the command line:

   ```
   $ mkdir telegraf
   ```

2. Then, create **`telegraf/telegraf.conf`** with the following lines:

   I. Influxdb : Configure the destination host of the data:

```
[[outputs.influxdb_v2]]
  urls = ["http://influxdb:8086"]
   organization = "myorg"
```

I. Configure the destination bucket to write into:

```
bucket = "iiot-book-2"
```

I. Finally, set the token for authentication on the previously defined **`influxInfluxDBdb`** :

```
token = "my-super-secret-auth-
```

```
token"
```

I. MQTT : Now, we can define the source of data and subscribe to the topics:

```
[[inputs.mqtt_consumer]]
  servers = ["tcp://mosquitto:1883"]
  topics = ["sensors/#"]
```

I. Mapping : Define the mapping between MQTT naming and destination on InfluxDB. The following code defines the name of the topic to publish data to and how to parse the measurement. The `"_"` wildcard means that we pass the name as is:

```
  data_format="value"
  data_type = "float"
  [[inputs.mqtt_consumer.topic_parsing]]
    topic = "sensors/_/_/_"
    measurement = "_/_/measurement/
_"
    tags = "_/device_name/_/field"
```

About the naming convention

In this book, I will not go into depth on the naming convention to keep things uncomplicated. However, it is important to define a naming convention for our measures from the beginning. There are two main tricks – carry out a name conversion directly on the edge (for example, we could rename all the

> *measures* **`<source/plant/office>.<measure>`***)*
>
> *or add a customer name prefix to the data broker –* **`<customer>.<source/plant/office>.<measure>`***.*

1. Configure the service : On Docker Compose, add the following service:

```
telegraf:
      container_name: telegraf
      image: telegraf
      depends_on:
        - influxdb
        - mosquitto
      volumes:
      - "./telegraf:/etc/telegraf"
```

   To allow interoperability among Docker Compose networks, add the following external network:

```
      networks:
      - iiot-book-2-net
```

To get a complete solution for the exercises, take a look at the repository at https://github.com/PacktPublishing/Hands-On-Industrial-Internet-of-

Things-Second-Edition .

Starting and testing the data flow

In the previous section, we config ured our data flow from MQTT to InfluxDB.

In this section, we want to start the data flow and inject data from the OPC-UA sample server, using Node-RED, to MQTT Mosquitto; we can reuse the edge client developed in *Chapter 6* . Follow these steps:

1. Start the data flow : From the directory where we configured Mosquitto InfluxDB and Telegraf, we can start all the services running:

```
$ docker compose up
```

2. Start the edge : From the directory where we configured Node-RED and OPC-UA sample server, we can start all the services running:

```
$ docker compose up
```

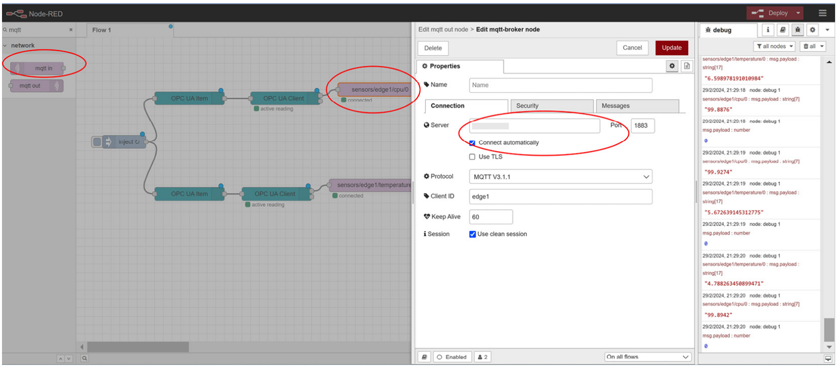
3. Show the services : From another command console, run the following command to see all the services up and running:

```
$ docker ps --format "table
{{.Names}}\t{{.Networks}}"
```

The expected result is as follows:

```
telegraf                  iiot-book-2-
net
dataflow-influxdb-1       iiot-book-2-
net
mosquitto                 iiot-book-2-
net
edge-nodered-1            edge_iiot-
book-2-net-edge
edge-opc-ua-server-1      edge_iiot-
book-2-net-edge
```

4. Configure NodeRED : Connect to http://localhost:1886 , the MQTT-out node to the flow defined in *Chapter 6* , and then the OPC-UA node. Double-click on the MQTT-out node, click on the pencil icon, and then configure the MQTT host server, as depicted in the following figure:


Node-RED
Deploy
mqtt
network
mqtt in
mqtt out
Flow 1
OPC UA Item
OPC UA Client
active reading
sensors/edge1/cpu/0
connected
inject
sensors/edge1/temperature
debug
all nodes
all
Delete
Cancel
Update
Properties
Connection
Security
Messages
Enabled
2
On all flows

Edit mqtt out node > Edit mqtt-broker node

Name: Name

Server: [redacted] Port: 1883

☑ Connect automatically

☐ Use TLS

Protocol: MQTT V3.1.1

Client ID: edge1

Keep Alive: 60

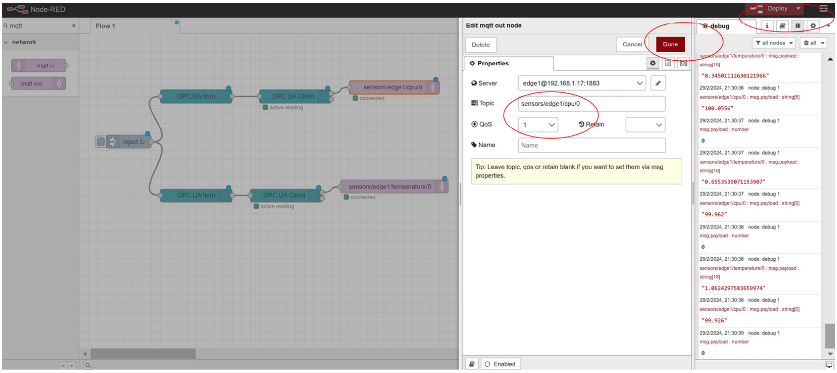
Session: ☑ Use clean session

```
sensors/edge1/temperature/0 : msg.payload :
string[17]
"6.598978191010984"
29/2/2024, 21:29:18   node: debug 1
sensors/edge1/cpu/0 : msg.payload : string[7]
"99.8876"
20/2/2024, 21:20:18   node: debug 1
msg.payload : number
0
29/2/2024, 21:29:19   node: debug 1
sensors/edge1/cpu/0 : msg.payload : string[7]
"99.9274"
29/2/2024, 21:29:19   node: debug 1
sensors/edge1/temperature/0 : msg.payload :
string[17]
"5.672639145312775"
29/2/2024, 21:29:19   node: debug 1
msg.payload : number
0
29/2/2024, 21:29:20   node: debug 1
sensors/edge1/temperature/0 : msg.payload :
string[17]
"4.788263450899471"
29/2/2024, 21:29:20   node: debug 1
sensors/edge1/cpu/0 : msg.payload : string[7]
"99.8942"
29/2/2024, 21:29:20   node: debug 1
msg.payload : number
0
```

Figure 8.2 – Configuring the MQTT-out node

The server's name is the public IP address of the MQTT data broker. If you start edge and data flow on the same host, you can use the local PC's IP address. Note that *the edge and the data flow network are not visible* .

Now, configure the MQTT topic to send data to **`sensors/edge1/cpu/0`** , as depicted in the following figure:


Node-RED
mqtt
network
mqtt in
mqtt out
Flow 1
inject ↻
OPC UA Item
OPC UA Client
active reading
sensors/edge1/cpu/0
connected
OPC UA Item
OPC UA Client
active reading
sensors/edge1/temperature/0
connected
Deploy
Delete
Cancel
Done
Properties
debug
all nodes
all
Enabled

Edit mqtt out node

Server: edge1@192.168.1.17:1883

Topic: sensors/edge1/cpu/0

QoS: 1

Retain:

Name: Name

Tip: Leave topic, qos or retain blank if you want to set them via msg properties.

```
sensors/edge1/temperature/0 : msg.payload :
string[19]
"0.34501112630121966"
29/2/2024, 21:30:36   node: debug 1
sensors/edge1/cpu/0 : msg.payload : string[8]
"100.0556"
29/2/2024, 21:30:37   node: debug 1
msg.payload : number
0
29/2/2024, 21:30:37   node: debug 1
sensors/edge1/temperature/0 : msg.payload :
string[18]
"0.6553539071153907"
29/2/2024, 21:30:37   node: debug 1
sensors/edge1/cpu/0 : msg.payload : string[6]
"99.962"
29/2/2024, 21:30:38   node: debug 1
msg.payload : number
0
29/2/2024, 21:30:38   node: debug 1
sensors/edge1/temperature/0 : msg.payload :
string[18]
"1.0624297583659974"
29/2/2024, 21:30:38   node: debug 1
sensors/edge1/cpu/0 : msg.payload : string[6]
"99.926"
29/2/2024, 21:30:39   node: debug 1
msg.payload : number
0
```

Figure 8.3 – Configuring the MQTT-out node

Repeat the same steps for the temperature measure.

1. Test on InfluxDB : Now, we can see the result of our job on InfluxDB. Connect to http://localhost:8086 , click on source , then explore the filters and submit to the graph viewer. The following figure shows the final results:

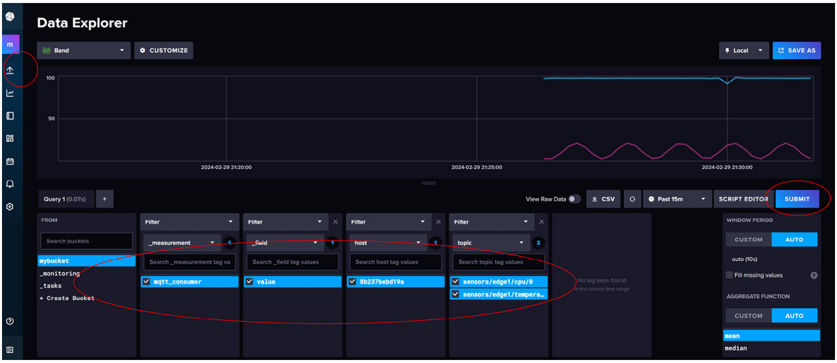

Data Explorer
Band
CUSTOMIZE
Local
SAVE AS
100
50
2024-02-29 21:20:00
2024-02-29 21:25:00
2024-02-29 21:30:00
Query 1 (0.07s)
+
View Raw Data
CSV
Past 15m
SCRIPT EDITOR
SUBMIT
FROM
Search buckets
mybucket
_monitoring
_tasks
+ Create Bucket
Filter
_measurement
1
Search _measurement tag va
☑ mqtt_consumer
Filter
_field
1
Search _field tag values
☑ value
Filter
host
1
Search host tag values
☑ 8b237bebd19a
Filter
topic
2
Search topic tag values
☑ sensors/edge1/cpu/0
☑ sensors/edge1/tempera…
No tag keys found
in the current time range
WINDOW PERIOD
CUSTOM
AUTO
auto (10s)
☐ Fill missing values
AGGREGATE FUNCTION
CUSTOM
AUTO
mean
median

Figure 8.4 – The graph of the time series just created on the InfluxDB Data Explorer

Visualizing data using Grafana

One very powerful tool to visualize data is Grafana . We will use Grafana also in *Chapter 9* . To install Grafana, just simply write the following **`docker-compose.yml`** file:

```
version: "3"
services:
  grafana:
    image: grafana/grafana-oss
    ports:
    - 3000:3000
    networks:
    - iiot-book-2-net
networks:
    iiot-book-2-net:
      external: true
```

From the command console on the same directory of the **`docker-compose.yml`** file, run the following command:

```
$ docker compose up
```

Now, we are ready to connect to Grafana. From the browser, connect to http://localhost:3000, using the username **`admin`** and password **`admin`** . A splash screen will require you to change your password; we can skip that for now and accept the default parameters, but it's recommended to change it in production mode.

The first step to connect to Grafana is to connect an InfluxDb database. From Connections , add an InfluxDB connection, as shown in the following figure:

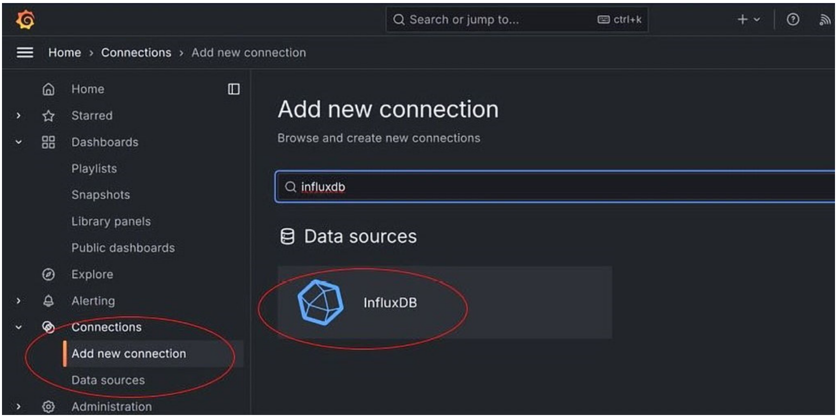
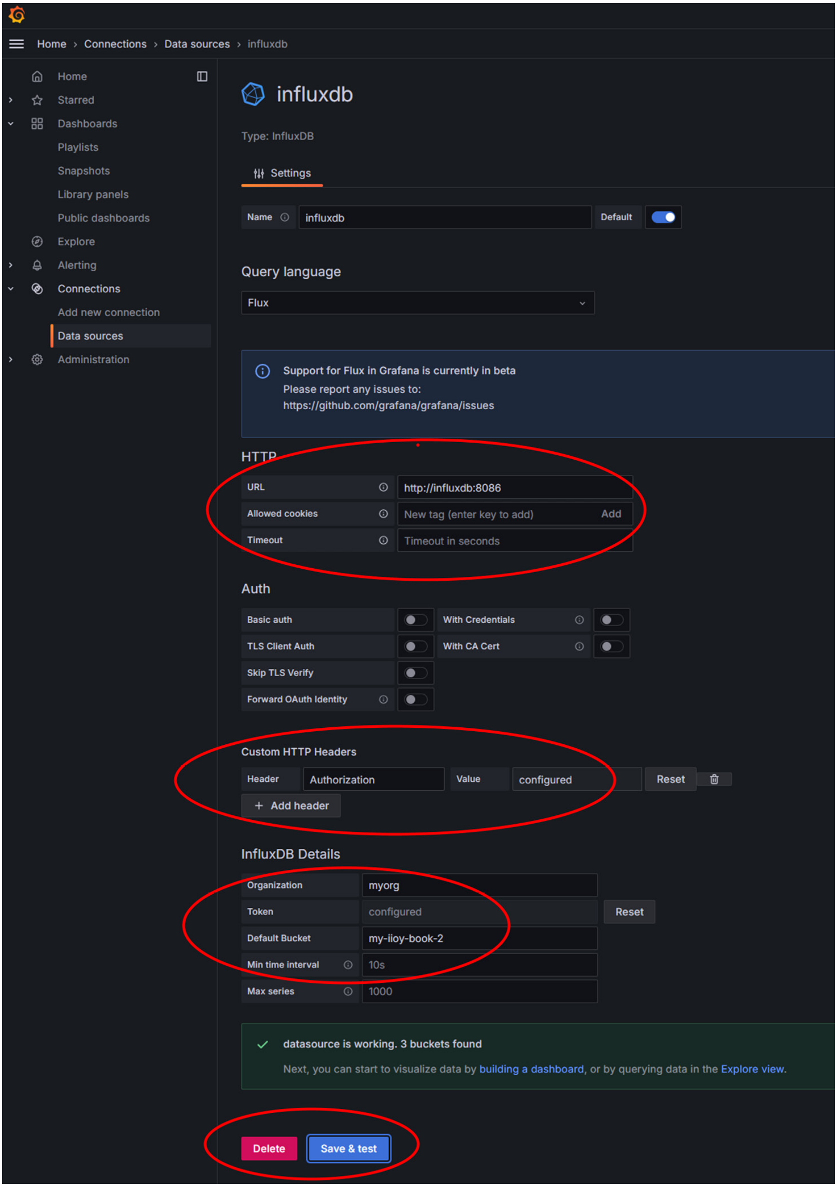

Search or jump to...
ctrl+k
Home › Connections › Add new connection
Home
Starred
Dashboards
Playlists
Snapshots
Library panels
Public dashboards
Explore
Alerting
Connections
Add new connection
Data sources
Administration
Add new connection
Browse and create new connections
influxdb
Data sources
InfluxDB

Figure 8.5 – Adding a new connection to InfluxDB

Now, we can configure the parameters of the connection:

- Name : `influxdb`
- Query language : Flux
- URL : `http://influxdb:8086`
- Custom HTTP Header :
    - Authorization : `token my-super-secret-auth-token`
- Organization : `myorg`
- Bucket : `iiot-book-2`

See the following figure for the details:


Home › Connections › Data sources › influxdb
Home
Starred
Dashboards
Playlists
Snapshots
Library panels
Public dashboards
Explore
Alerting
Connections
Add new connection
Data sources
Administration
Settings
Add header
Reset
Delete
Save & test

influxdb

Type: InfluxDB

Name: influxdb

Default: on

Query language: Flux

Support for Flux in Grafana is currently in beta
Please report any issues to:
https://github.com/grafana/grafana/issues

HTTP

URL: http://influxdb:8086

Allowed cookies: New tag (enter key to add) Add

Timeout: Timeout in seconds

Auth

Basic auth: off | With Credentials: off

TLS Client Auth: off | With CA Cert: off

Skip TLS Verify: off

Forward OAuth Identity: off

Custom HTTP Headers

Header: Authorization

Value: configured

InfluxDB Details

Organization: myorg

Token: configured

Default Bucket: my-iioy-book-2

Min time interval: 10s

Max series: 1000

datasource is working. 3 buckets found

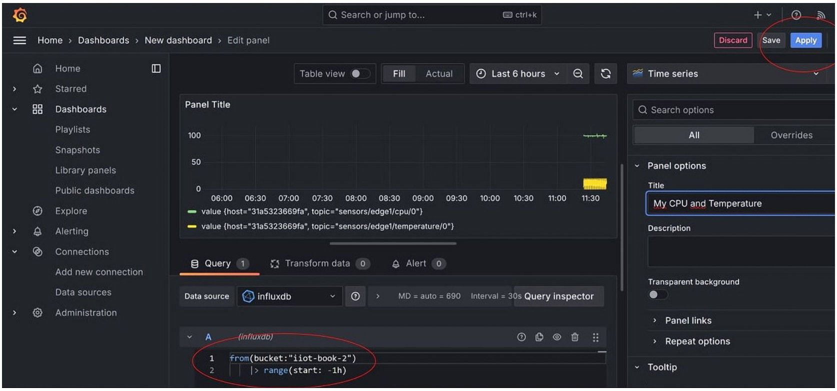
Next, you can start to visualize data by building a dashboard, or by querying data in the Explore view.

Figure 8.6 – Configuring a new connection to InfluxDB

Now, we can add a new dashboard. From the upper-left corner (*Figure 8 .7*), add a new dashboard, and in the query area, add the following query:

```
from(bucket:"iiot-book-2")
    |> range(start: -1h)
```

Finally, in the upper-right corner, click on Apply :


Search or jump to...
ctrl+k
Home › Dashboards › New dashboard › Edit panel
Discard
Save
Apply
Home
Starred
Dashboards
Playlists
Snapshots
Library panels
Public dashboards
Explore
Alerting
Connections
Add new connection
Data sources
Administration
Table view
Fill
Actual
Last 6 hours
Time series
Panel Title
100
50
0
06:00
06:30
07:00
07:30
08:00
08:30
09:00
09:30
10:00
10:30
11:00
11:30
Search options
All
Overrides
Panel options
Title
Description
Transparent background
Panel links
Repeat options
Tooltip
Query 1
Transform data 0
Alert 0
Data source
influxdb
MD = auto = 690
Interval = 30s
Query inspector
A
(influxdb)

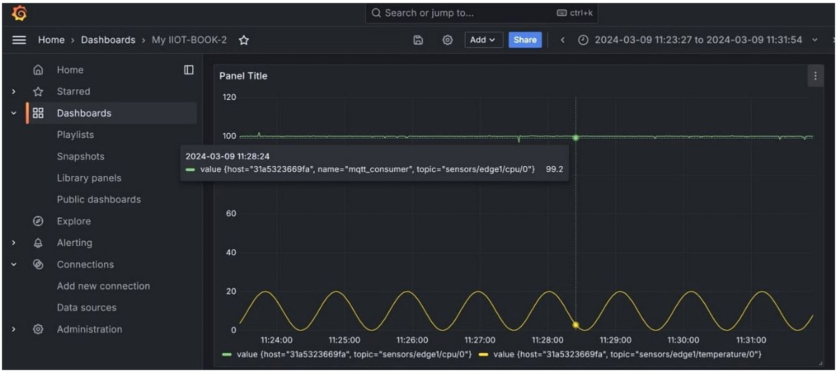
Title: My CPU and Temperature

value {host="31a5323669fa", topic="sensors/edge1/cpu/0"}
value {host="31a5323669fa", topic="sensors/edge1/temperature/0"}

```
from(bucket:"iiot-book-2")
  |> range(start: -1h)
```

Figure 8.7 – Configuring a Grafana dashboard

Voilà! Your dashboard is now ready.


Search or jump to...
ctrl+k
Home › Dashboards › My IIOT-BOOK-2
Add
Share
2024-03-09 11:23:27 to 2024-03-09 11:31:54
Home
Starred
Dashboards
Playlists
Snapshots
Library panels
Public dashboards
Explore
Alerting
Connections
Add new connection
Data sources
Administration
Panel Title
120
100
60
40
20
0
11:24:00
11:25:00
11:26:00
11:27:00
11:28:00
11:29:00
11:30:00
11:31:00

2024-03-09 11:28:24
value {host="31a5323669fa", name="mqtt_consumer", topic="sensors/edge1/cpu/0"} 99.2

value {host="31a5323669fa", topic="sensors/edge1/cpu/0"}
value {host="31a5323669fa", topic="sensors/edge1/temperature/0"}

Figure 8.8 – The final Grafana dashboard

To get a complete solution for the exercises, take a look at the repository at https://github.com/PacktPublishing/Hands-On-Industrial-Internet-of-Things-Second-Edition .

Developing our batch analytics with Airflow

In an IIoT framework, we need fast data-processing analytics (a hot path) and a long-running batch processor to develop our digital twin or deep learning analytics (a cold path). In this section, we want to experiment with it using a batch-processing analytics platform called Apache Airflow . Airflow has good support for the workflow, along with support for a directed acyclic graph (DAG), plugins, and the connector.

> ***Airflow is supported on the cloud***

> *Airflow can be easily installed on the most important cloud platforms. AWS and GCP support Airflow natively.*

Installing Airflow for production

Installing and running the Apache Airflow single node is quite simple. The fastest way is to use **`docker-compose.yml`** , provided on the official GitHub repository of this book:

1. Prepare the directories : Create the following directories:

   ```
   $ mkdir ./airflow
   $ cd ./airflow
   $ mkdir ./dags
   $ mkdir ./logs
   $ mkdir ./plugins
   $ mkdir ./config
   ```

   On **`.env`** , add the following lines to configure the default user and add support for InfluxDB:

   ```
   AIRFLOW_UID=50000
   _PIP_ADDITIONAL_REQUIREMENTS=apache-
   airflow-providers-influxdb
   ```

2. Build Docker Compose : Download the **`docker-compose.yml`** file from https://airflow.apache.org/docs/apache-airflow/2.8.1/docker-compose.yaml .

Add, also the following portion of code for each service to allow service interoperability between InfluxDB and Airflow:

```
    networks:
      - iiot-book-2-net-central
```

3. Add a network for connection with InfluxDB : Add the following to the end of **`docker-compose.yml`** :

```
networks:
    iiot-book-2-net-central:
      external: true
```

To get a complete solution for the exercises, take a look at the repository at https://github.com/PacktPublishing/Hands-On-Industrial-Internet-of-Things-Second-Edition .

4. Initialize the network and database : Now, we can start the Airflow scheduler web server and trigger:

```
$ docker network create iiot-book-2-net-central
$ docker compose up airflow-init
```

5. Start/stop the services : Then, start the services. Executing the following from the command line:

```
$ docker compose up
```

Airflow takes a few minutes to start up , so be patient. Docker will also start the UI (see the *Airflow UI* section of the `docker-compose.yml`).

To check the list of services, run the following:

```
$ docker ps --format '{{.Names}}'
```

The output should be like the following:

```
airflow-airflow-triggerer-1
airflow-airflow-worker-1
airflow-airflow-scheduler-1
airflow-airflow-webserver-1
airflow-postgres-1
airflow-redis-1
```

To close all services, use the following:

```
$ docker compose down --volumes --rmi
all
```

Installing Airflow on a base host (installation for development)

In the following exercise, we will use the Docker version. However, if you do not have enough memory or want to debug your analytics, you can install Airflow using python's `pip install` . Follow these steps:

1. Install Python 3.
2. Prepare the directories : From the command console, run the following commands:

```
$ mkdir ./dags
$ mkdir ./logs
$ mkdir ./plugins
$ mkdir ./config
$ pip install apache-airflow apache-
airflow-providers-influxdb
```

3. Export `AIRFLOW_HOME` with the following commands, based on the OS:
   - *On Windows* , this is done as follows:

```
$ set AIRFLOW_HOME=<current
directory>
```

   - *On macOS or Linux* , this is done as follows:

```
$ export AIRFLOW_HOME=$(pwd)
```

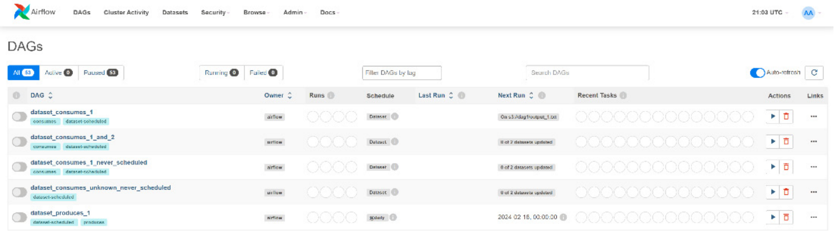
6. Start/stop the services : We are now ready to start Airflow. From the

command console, run this command:

```
$ airflow standalone
```

Connecting to the Airflow UI

We can now open the browser to http://localhost:8080 with `airflow` as the username and password:


Airflow
DAGs
Cluster Activity
Datasets
Security
Browse
Admin
Docs
21:03 UTC
DAGs
All 53
Active 0
Paused 53
Running 0
Failed 0
Filter DAGs by tag
Search DAGs
Auto-refresh

DAG	Owner	Runs	Schedule	Last Run	Next Run	Recent Tasks	Actions	Links
dataset_consumes_1 consumes dataset-scheduled	airflow		Dataset		On s3://dag1/output_1.txt			
dataset_consumes_1_and_2 consumes dataset-scheduled	airflow		Dataset		0 of 2 datasets updated			
dataset_consumes_1_never_scheduled consumes dataset-scheduled	airflow		Dataset		0 of 2 datasets updated			
dataset_consumes_unknown_never_scheduled dataset-scheduled	airflow		Dataset		0 of 2 datasets updated			
dataset_produces_1 dataset-scheduled produces	airflow		@daily		2024 02 16, 00:00:00			

Figure 8.9 – The Airflow UI

Airflow architecture and concepts

A directed graph is any graph whose vertices and edges have a specific order or direction. A directed graph without cycles is called a DAG. A cycle is a collection of vertices connected, forming an unconstrained chain. Airflow is a powerful engine based on a DAG. *Figure 8 .10* shows how a DAG works:

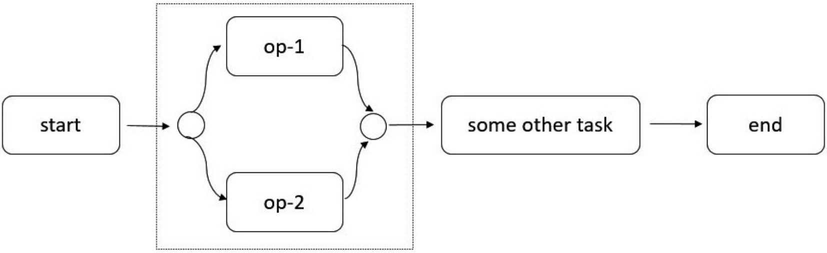


Figure 8.10 – An Airflow DAG

In short, a DAG defines a workflow of tasks; each task can support an operator. The most famous operator is the *Python operator* , but you can define your custom operator to support non-native Python analytics.

Airflow architecture consists of separate parts:

- Scheduler is the orchestrator of Airflow. Scheduler is responsible for parsing DAG files, managing database state, and scheduling tasks.
- Webserver is the web interface of Airflow that allows you to manage DAGs.
- Worker is a Celery worker application that consumes and executes tasks scheduled by Scheduler when using a Celery-like executor. It is possible to have many workers (scaling among a Kubernetes Pod, Docker, and VMs).
- Airflow meta-database , which is used to store all information about

DAGs and tasks; on the current Docker infrastructure, we used PostgreSQL and REDIS.

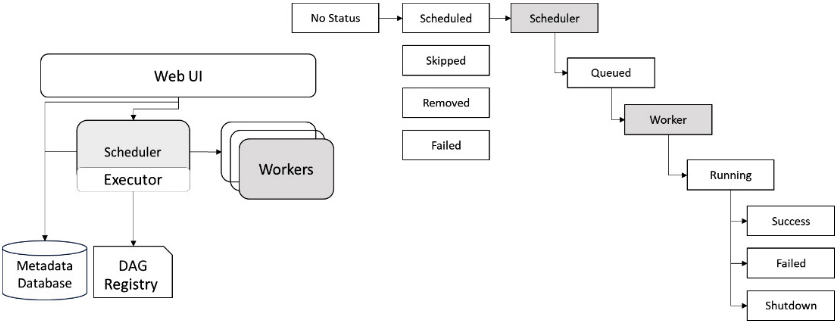

No Status
Scheduled
Scheduler
Skipped
Removed
Failed
Queued
Worker
Running
Success
Failed
Shutdown
Web UI
Scheduler
Executor
Workers
Metadata Database
DAG Registry

Figure 8.11 – Airflow architecture and DAG lifecycle

When an analytic is scheduled, the executor pushes it into a queue, and a worker can run it. If the analytic fails, Airflow restarts it. If the analytic executes the task successfully, it will be scheduled in accordance with the trigger policies.

The first analytic on Airflow

Airflow analytics are purely Python analytics, so code configuration and scheduling can be done in a single Python file. To build an analytic on Airflow, we need to create a Python file called

`loggingdag.py` in the previously created **`dags`** folder.

Now, add **`python import`**:

```
import datetime
from airflow.models.dag import DAG
from airflow.operators.python import
PythonOperator
from airflow.utils.dates import
days_ago
```

Define the core main function – in this case, a simple logging function:

```
def print_context(**kwargs):
    """Print the Airflow context."""
    import logging
    logging.info("=>===========<=")
    logging.info(kwargs)
    return "My first DAG"
```

Now, we can schedule the DAG every minute, and define the name, the owner, and the flow:

```
with DAG('loggingdag',
description='Simple Logging DAG',
    default_args = {'owner': 'iiot-
book2'},
    start_date=days_ago(0),
    schedule_interval='*/1 * * * *')
as dag:
```

We define ahead two tasks, executing the Python operator; the Python operators are going to call the main function defined previously:

```
    t1 =
PythonOperator(task_id="print_the_context",
                   provide_context=True,
                    python_callable=print_c
    t2 =
PythonOperator(task_id="print_without_conte
                python_callable=print_conte
```

Finally, we can say that the first task follows the second task:

```
    t1 >> t2
```

The `>>` and `<<` python characters allow us to define the task dependencies within a DAG. For example, **`task1 >> [task2, task3]`** means that **`task2`** and **`task3`** will be run after **`task1`**.

We are ready to see Airflow in action; connect to http://localhost:8080 with **`airflow`** as the username and password, and then activate the DAG, as depicted in the following figure:

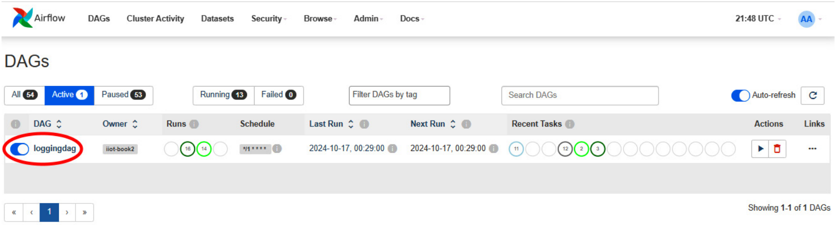

Airflow
DAGs
Cluster Activity
Datasets
Security
Browse
Admin
Docs
21:48 UTC
AA
DAGs
All 54
Active 1
Paused 53
Running 13
Failed 0
Filter DAGs by tag
Search DAGs
Auto-refresh
DAG
Owner
Runs
Schedule
Last Run
Next Run
Recent Tasks
Actions
Links
loggingdag
iiot-book2
16
14
*/1 * * * *
2024-10-17, 00:29:00
2024-10-17, 00:29:00
11
12
2
3
Showing 1-1 of 1 DAGs

Figure 8.12 – Activating the analytic

Now, we can click on the name **`loggingdag`** and then on Graph to see the flow. The following figure shows the flow.

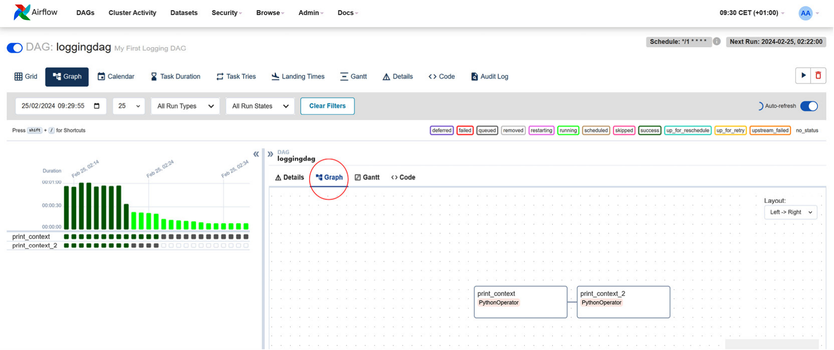
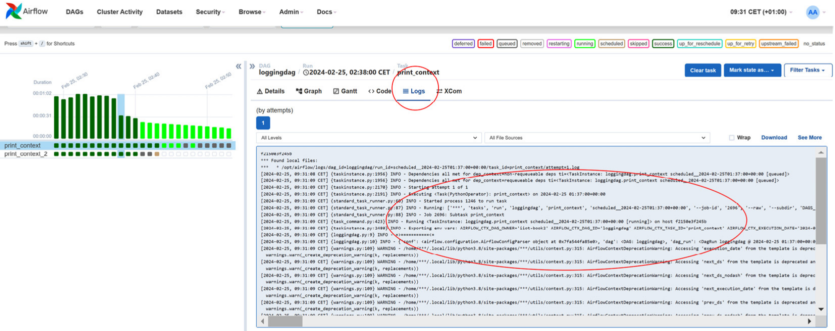

Airflow
DAGs
Cluster Activity
Datasets
Security
Browse
Admin
Docs
09:30 CET (+01:00)
AA
DAG: loggingdag
My First Logging DAG
Schedule: */1 * * * *
Next Run: 2024-02-25, 02:22:00
Grid
Graph
Calendar
Task Duration
Task Tries
Landing Times
Gantt
Details
Code
Audit Log
25/02/2024 09:29:55
25
All Run Types
All Run States
Clear Filters
Auto-refresh
Press shift + / for Shortcuts
deferred
failed
queued
removed
restarting
running
scheduled
skipped
success
up_for_reschedule
up_for_retry
upstream_failed
no_status
Duration
00:01:00
00:00:30
00:00:00
Feb 25, 02:14
Feb 25, 02:24
Feb 25, 02:34
print_context
print_context_2
DAG
loggingdag
Details
Graph
Gantt
Code
Layout:
Left -> Right
print_context
PythonOperator
print_context_2
PythonOperator

Figure 8.13 – Showing the flow

Finally, click on the left panel and then the Logs tab to see the log for the specific execution:


Airflow
DAGs
Cluster Activity
Datasets
Security
Browse
Admin
Docs
09:31 CET (+01:00)
AA
Press shift + / for Shortcuts
deferred
failed
queued
removed
restarting
running
scheduled
skipped
success
up_for_reschedule
up_for_retry
upstream_failed
no_status
Duration
00:01:02
00:00:31
00:00:00
Feb 25, 02:30
Feb 25, 02:40
Feb 25, 02:50
print_context
print_context_2
DAG
loggingdag
Run
2024-02-25, 02:38:00 CET
Task
print_context
Clear task
Mark state as...
Filter Tasks
Details
Graph
Gantt
Code
Logs
XCom
(by attempts)
1
All Levels
All File Sources
Wrap
Download
See More

```
f2150e3f245b
*** Found local files:
***   * /opt/airflow/logs/dag_id=loggingdag/run_id=scheduled__2024-02-25T01:37:00+00:00/task_id=print_context/attempt=1.log
[2024-02-25, 09:31:08 CET] {taskinstance.py:1956} INFO - Dependencies all met for dep_context=non-requeueable deps ti=<TaskInstance: loggingdag.print_context scheduled__2024-02-25T01:37:00+00:00 [queued]>
[2024-02-25, 09:31:09 CET] {taskinstance.py:1956} INFO - Dependencies all met for dep_context=requeueable deps ti=<TaskInstance: loggingdag.print_context scheduled__2024-02-25T01:37:00+00:00 [queued]>
[2024-02-25, 09:31:09 CET] {taskinstance.py:2170} INFO - Starting attempt 1 of 1
[2024-02-25, 09:31:09 CET] {taskinstance.py:2191} INFO - Executing <Task(PythonOperator): print_context> on 2024-02-25 01:37:00+00:00
[2024-02-25, 09:31:09 CET] {standard_task_runner.py:60} INFO - Started process 1246 to run task
[2024-02-25, 09:31:09 CET] {standard_task_runner.py:87} INFO - Running: ['***', 'tasks', 'run', 'loggingdag', 'print_context', 'scheduled__2024-02-25T01:37:00+00:00', '--job-id', '2696', '--raw', '--subdir', 'DAGS_
[2024-02-25, 09:31:09 CET] {standard_task_runner.py:88} INFO - Job 2696: Subtask print_context
[2024-02-25, 09:31:09 CET] {task_command.py:423} INFO - Running <TaskInstance: loggingdag.print_context scheduled__2024-02-25T01:37:00+00:00 [running]> on host f2150e3f245b
[2024-02-25, 09:31:09 CET] {taskinstance.py:2480} INFO - Exporting env vars: AIRFLOW_CTX_DAG_OWNER='iiot-book2' AIRFLOW_CTX_DAG_ID='loggingdag' AIRFLOW_CTX_TASK_ID='print_context' AIRFLOW_CTX_EXECUTION_DATE='2024-0
[2024-02-25, 09:31:09 CET] {loggingdag.py:9} INFO - =>===========<=
[2024-02-25, 09:31:09 CET] {loggingdag.py:10} INFO - {'conf': <airflow.configuration.AirflowConfigParser object at 0x7fa544fa85e0>, 'dag': <DAG: loggingdag>, 'dag_run': <DagRun loggingdag @ 2024-02-25 01:37:00+00:0
[2024-02-25, 09:31:09 CET] {warnings.py:109} WARNING - /home/***/.local/lib/python3.8/site-packages/***/utils/context.py:315: AirflowContextDeprecationWarning: Accessing 'execution_date' from the template is deprec
  warnings.warn(_create_deprecation_warning(k, replacements))
[2024-02-25, 09:31:09 CET] {warnings.py:109} WARNING - /home/***/.local/lib/python3.8/site-packages/***/utils/context.py:315: AirflowContextDeprecationWarning: Accessing 'next_ds' from the template is deprecated an
  warnings.warn(_create_deprecation_warning(k, replacements))
[2024-02-25, 09:31:09 CET] {warnings.py:109} WARNING - /home/***/.local/lib/python3.8/site-packages/***/utils/context.py:315: AirflowContextDeprecationWarning: Accessing 'next_ds_nodash' from the template is deprec
  warnings.warn(_create_deprecation_warning(k, replacements))
[2024-02-25, 09:31:09 CET] {warnings.py:109} WARNING - /home/***/.local/lib/python3.8/site-packages/***/utils/context.py:315: AirflowContextDeprecationWarning: Accessing 'next_execution_date' from the template is d
  warnings.warn(_create_deprecation_warning(k, replacements))
[2024-02-25, 09:31:09 CET] {warnings.py:109} WARNING - /home/***/.local/lib/python3.8/site-packages/***/utils/context.py:315: AirflowContextDeprecationWarning: Accessing 'prev_ds' from the template is deprecated an
  warnings.warn(_create_deprecation_warning(k, replacements))
```

Figure 8.14 – Logging – the final result

Connecting Airflow to InfluxDB

In this exercise, we want to build our first anomaly detection algorithm to detect potential issues with CPU usage:

1. Configure an InfluxDB connection : The first step is to configure an InfluxDB connection on Airflow. From the Airflow UI, click on Admin | Connections and add a new connection, as shown in the following figure:

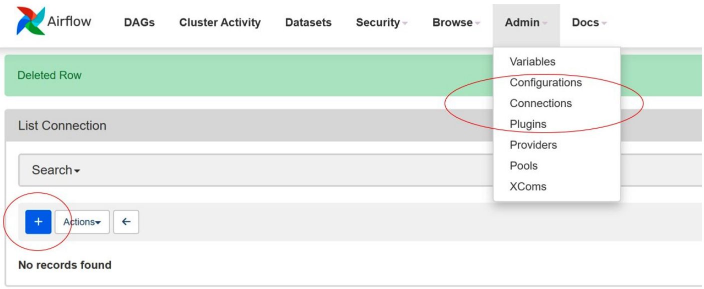
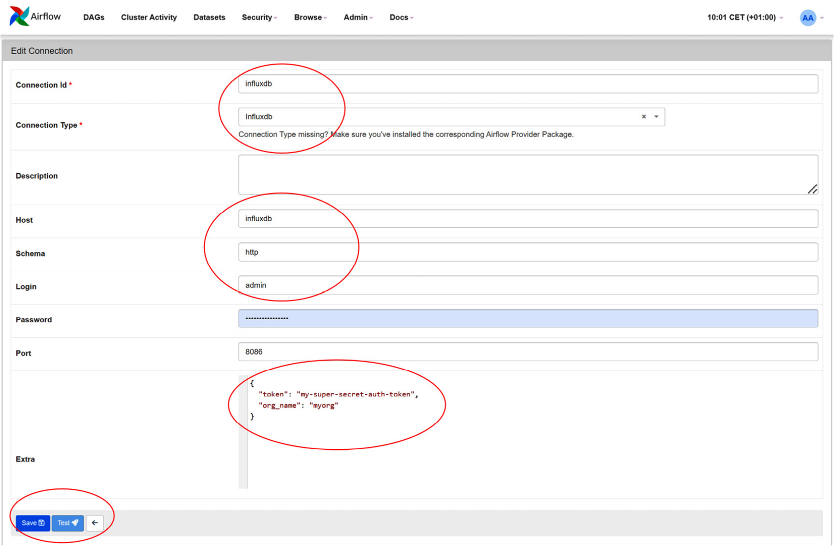

Airflow
DAGs
Cluster Activity
Datasets
Security
Browse
Admin
Docs
Deleted Row
List Connection
Search
+
Actions
←
No records found
Variables
Configurations
Connections
Plugins
Providers
Pools
XComs

Figure 8.15 – Adding a connection on Airflow

Set the connection name to `influxdb` , select influxdb as the connection type, set the hostname to `influxdb` , set the port to `8086` , and set the schema to `http` . Now, we need to set the organization and token as follows:

```
{
  "token": "my-super-secret-auth-token",
  "org_name": "myorg"
}
```

The final results should appear as follows:


Airflow
DAGs
Cluster Activity
Datasets
Security
Browse
Admin
Docs
10:01 CET (+01:00)
AA
Edit Connection
Save
Test
←

Connection Id * influxdb

Connection Type * Influxdb

Connection Type missing? Make sure you've installed the corresponding Airflow Provider Package.

Description

Host influxdb

Schema http

Login admin

Password ••••••••••••••••

Port 8086

Extra

```
{
  "token": "my-super-secret-auth-token",
  "org_name": "myorg"
}
```

Figure 8.16 – Configuring an InfluxDB connection

These parameters are the same as we defined on the Docker Compose file of InfluxDB.

1. Build an anomaly detection analytic : We are now ready to build and run our analytics. In the **`dags`** folder, create a file with the name **`anomalydag.py`**.
2. Define the packages to import:

   ```
   import datetime
   from airflow.models.dag import DAG
   from airflow.operators.python import
   PythonOperator
   from airflow.utils.dates import
   days_ago
   ```

3. Add one more package for InfluxDB:

   ```
   from
   airflow.providers.influxdb.hooks.influxd
   import InfluxDBHook
   ```

4. Define the main function:

   ```
   def ad_by_quantile():
       import numpy as np
       import logging
   ```

5. Inside the function, build an anomaly detection model by percentile as an inner class:

```
class SimplePercentileAD:
        def fit(self, X, y=None):
            percentile_range = (10,
90)
            self.p= np.percentile(X,
percentile_range)
            return self
        def predict(self, X):
            anomalies = np.where((X
< self.p[0]) |
                                 (X
> self.p[1]))
            return anomalies
```

The model is very simple and uses the scikit-learn model estimator (https://scikit-learn.org/stable/modules/generated/sklearn.base.BaseEstimator.html) to replicate the same mechanism, on **`fit`** , you train the model, and on **`predict`** , you perform a test. Familiarize yourself with this concept, as we will explore it in detail when we study how artificial intelligence can be applied to IIoT in *Chapters 11* , *12* , and *13* . This model is a statistical one that learns boundaries from the training dataset; then, it tests whether the current dataset is beyond the calculated limits. In other words, it is a simple rule-based algorithm where the upper and lower limit is automatically calculated:

1. Now, we can instantiate the model:

```
      model = SimplePercentileAD()
```

2. Define a variable about the sensor name:

```
      SENSOR_NAME='sensors/edge1/
cpu/0'
```

3. Define the InfluxDB SQL query:

```
SQL = '''
from(bucket: "iiot-book-2")
  |> range(start: {start}, stop:
{stop})
  |> filter(fn: (r) => r["topic"] ==
"{sensor_name}")
  |> aggregateWindow(every: 1m, fn:
mean, createEmpty: true)
  |> yield(name: "mean")'''
```

This query is a simple template query where we ask for data from a sensor in a specific period range (from `start` to `stop`). Samples will be interpolated every minute.

4. Now, we can use our query to *train* our model on the last three hours:

```
    df =
InfluxDBHook(conn_id="influxdb").query_t
stop="-5m",
sensor_name=SENSOR_NAME))
    model.fit(df['_value'].values)
```

5. Finally, we can *test* the last five minutes of data:

```
    df =
InfluxDBHook(conn_id="influxdb").query_t
stop="0m", sensor_name=SENSOR_NAME))
    anomalies =
model.predict(df['_value'].values)
```

Training and testing on machine learning

In machine learning during training, we learn the patterns and the algorithm, and then we identify the rules and the correlation between input (**`X`** *) and output (* **`y`** *). During a test or prediction, we evaluate the current dataset to identify anomalies, predict data, or classify the input. The training phase is, normally, a very long step, so we do not train and test at the same time. In this exercise, we will train on a large dataset, and then we will test on the last five minutes of data, but we should split this step into two different phases. Training should be done rarely – for example, every day. Testing is fast and can be done as needed. We will see it in Chapter 14 . To achieve this goal, you can practice defining where to train another DAG, and then save the model using pickle to share the model among the two DAGs. During the test phase,*

> *DAG just recovers the model with* `pickle.load` *and uses it to perform the prediction.*

Now, we are ready to prepare our report:

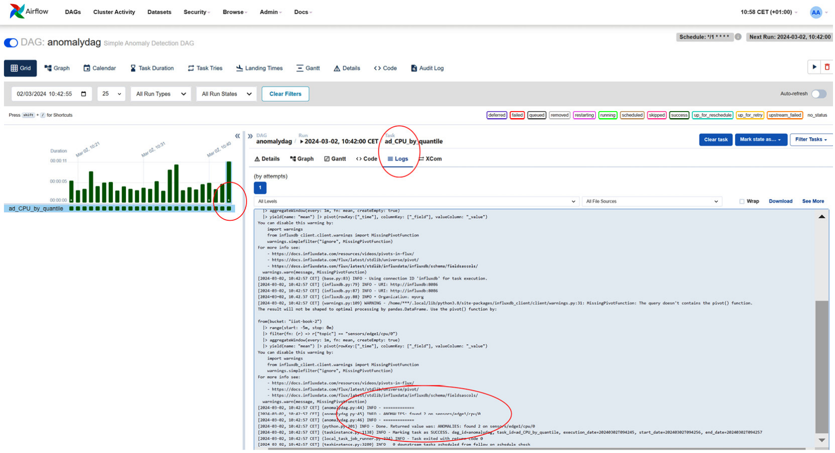
```
    ret = f"ANOMALIES: found
{np.sum(anomalies)} on {SENSOR_NAME}"
    logging.info("=============")
    logging.info(ret)
    logging.info("=============")
```

The report is just a simple log that you can see on the Logs panel of Airflow. The last piece of code defines the DAG and schedules it:

```
with DAG('anomalydag',
description='Simple Anomaly Detection
DAG',
    default_args = {'owner': 'iiot-
book2'},
    start_date=days_ago(0),
    schedule_interval='*/1 * * * *')
as dag:
    t1 =
PythonOperator(task_id="ad_CPU_by_quantile"
                    provide_context=True,
                    python_callable=ad_by_q
    t1
```

We do not need a description of the previous code because we have already explored how Airflow works.

Finally, from the Airflow UI, click on anomalydag and open the Logs panel of the last run:


Airflow
DAGs
Cluster Activity
Datasets
Security
Browse
Admin
Docs
10:58 CET (+01:00)
AA
DAG: anomalydag Simple Anomaly Detection DAG
Schedule: */1 * * * *
Next Run: 2024-03-02, 10:42:00
Grid
Graph
Calendar
Task Duration
Task Tries
Landing Times
Gantt
Details
Code
Audit Log
02/03/2024 10:42:55
25
All Run Types
All Run States
Clear Filters
Auto-refresh
Press shift + / for Shortcuts
deferred
failed
queued
removed
restarting
running
scheduled
skipped
success
up_for_reschedule
up_for_retry
upstream_failed
no_status
Duration
00:00:11
00:00:05
00:00:00
Mar 02, 10:21
Mar 02, 10:31
Mar 02, 10:40
ad_CPU_by_quantile
DAG
anomalydag
Run
2024-03-02, 10:42:00 CET
Task
ad_CPU_by_quantile
Clear task
Mark state as...
Filter Tasks
Details
Graph
Gantt
Code
Logs
XCom
(by attempts)
1
All Levels
All File Sources
Wrap
Download
See More

```
  |> aggregateWindow(every: 1m, fn: mean, createEmpty: true)
  |> yield(name: "mean") |> pivot(rowKey:["_time"], columnKey: ["_field"], valueColumn: "_value")
You can disable this warning by:
    import warnings
    from influxdb_client.client.warnings import MissingPivotFunction
    warnings.simplefilter("ignore", MissingPivotFunction)
For more info see:
    - https://docs.influxdata.com/resources/videos/pivots-in-flux/
    - https://docs.influxdata.com/flux/latest/stdlib/universe/pivot/
    - https://docs.influxdata.com/flux/latest/stdlib/influxdata/influxdb/schema/fieldsascols/
  warnings.warn(message, MissingPivotFunction)
[2024-03-02, 10:42:57 CET] {base.py:83} INFO - Using connection ID 'influxdb' for task execution.
[2024-03-02, 10:42:57 CET] {influxdb.py:79} INFO - URI: http://influxdb:8086
[2024-03-02, 10:42:57 CET] {influxdb.py:87} INFO - URI: http://influxdb:8086
[2024-03-02, 10:42:57 CET] {influxdb.py:88} INFO - Organization: myorg
[2024-03-02, 10:42:57 CET] {warnings.py:109} WARNING - /home/***/.local/lib/python3.8/site-packages/influxdb_client/client/warnings.py:31: MissingPivotFunction: The query doesn't contains the pivot() function.
The result will not be shaped to optimal processing by pandas.DataFrame. Use the pivot() function by:

from(bucket: "iiot-book-2")
  |> range(start: -5m, stop: 0m)
  |> filter(fn: (r) => r["topic"] == "sensors/edge1/cpu/0")
  |> aggregateWindow(every: 1m, fn: mean, createEmpty: true)
  |> yield(name: "mean") |> pivot(rowKey:["_time"], columnKey: ["_field"], valueColumn: "_value")
You can disable this warning by:
    import warnings
    from influxdb_client.client.warnings import MissingPivotFunction
    warnings.simplefilter("ignore", MissingPivotFunction)
For more info see:
    - https://docs.influxdata.com/resources/videos/pivots-in-flux/
    - https://docs.influxdata.com/flux/latest/stdlib/universe/pivot/
    - https://docs.influxdata.com/flux/latest/stdlib/influxdata/influxdb/schema/fieldsascols/
  warnings.warn(message, MissingPivotFunction)
[2024-03-02, 10:42:57 CET] {anomalydag.py:44} INFO - =============
[2024-03-02, 10:42:57 CET] {anomalydag.py:45} INFO - ANOMALIES: found 2 on sensors/edge1/cpu/0
[2024-03-02, 10:42:57 CET] {anomalydag.py:46} INFO - =============
[2024-03-02, 10:42:57 CET] {python.py:201} INFO - Done. Returned value was: ANOMALIES: found 2 on sensors/edge1/cpu/0
[2024-03-02, 10:42:57 CET] {taskinstance.py:1138} INFO - Marking task as SUCCESS. dag_id=anomalydag, task_id=ad_CPU_by_quantile, execution_date=20240302T094245, start_date=20240302T094256, end_date=20240302T094257
[2024-03-02, 10:42:57 CET] {local_task_job_runner.py:234} INFO - Task exited with return code 0
[2024-03-02, 10:42:57 CET] {taskinstance.py:3280} INFO - 0 downstream tasks scheduled from follow on schedule check
```

Figure 8.17 – Showing the event

Airflow supports several methods for notification (e.g., SNS, SQS, and SMTP), but we will customize our own method, implementing `BaseNotifier`.

Developing our online analytics with Airflow

Triggering an analytic at a specific time is the most common path. In an IIoT scenario, however, we could need to process analytics immediately. These analytics can be very simple and just involve checking value excursion (EMA). We differentiated between these two scenarios earlier when referring to a cold path versus a hot path. Feel free to refer to *Figure 8 .1* to refresh your memory on the two terms.

To build an online analytic working on a hot path, we need to connect Airflow directly to the MQTT data broker. The first step is developing the analytic on airflow, and then we can build a simple subscriber that can call Airflow when a new data point is received. Let's see how:

1. Develop a simple excursion monitoring analytic. The analytic receives two parameters – the threshold to test and the value.

   Import the necessary packages:

   ```
   import datetime
   import json
   ```

```
from airflow.models.dag import DAG
from airflow.operators.python import
PythonOperator
from airflow.utils.dates import
days_ago
```

2. Write a simple function that checks whether the **`value`** param is greater than the **`threshold`** param. When **`value`** exceeds **`threshold`**, we can raise a simple log message:

```
def test_rule(**kwargs):
    import logging
    params = kwargs['params']
    source = params['source']
    value = params['value']
    threshold = params['params']
["threshold"]
    ret = "Success without issue"
    if float(value) >=
float(threshold):
        ret = f"Anomaly {value} >
{threshold} on {source}"
        logging.info(ret)
    return ret
```

3. Finally, define the DAG, but schedule it just once because we do not need to schedule it repetitively, only upon the availability of data:

```
with DAG('ruledag', description='My
First Rule DAG',
    default_args = {'owner': 'iiot-
book2'},
    start_date=days_ago(0),
    schedule_interval='@once') as
dag:
    t1 =
PythonOperator(task_id="test_rule",
                             provide_context=True,
                              python_callable=test
```

4. Implement a simple MQTT subscriber that calls Airflow. To call Airflow from externally, we can use an HTTP **`POST`** request such as the following:

```
curl -X POST "http://airflow:8080/
api/v1/dags/ruledag/dagRuns" -
H  "accept: application/json" -
H  "Content-Type: application/json"
-d '{"conf":{}}' --user
"airflow:airflow"
```

5. We will develop this service in Python. Before coding, define a simple **`config.json`** file with the following content:

```
{
    "sensors/edge1/cpu/0": [
        {"ruledag":
{"threshold":99}}
    ]
}
```

What did we do in *step 5* ? The key is the MQTT topic to subscribe to; the value contains the name of the analytic and a **`threshold`** param.

6. We are going to use the **`paho-mqtt`** package, so install it using the following:

```
$ pip install requests paho-mqtt
```

7. Now, we can build our **`airflow-mqtt caller`**. Create a Python file named **`main.py`**. Import the packages:

```
import json
import paho.mqtt.client as mqtt
import logging
import os
import requests
import asyncio
```

8. Define a simple **`STATIC`** class to share the configuration. Do not forget that a subscriber works on a separate thread:

```
class STATIC:
    mapping = {}
    airflow_endpoint = None
    airflow_username = None
    airflow_password = None
```

9. When connecting to an MQTT server, we subscribe to all topics:

```
# The callback for when the client
```

```
receives a CONNACK response from the
server.
def on_connect(client, userdata,
flags, reason_code, properties):
    print(f"Connected with result
code {reason_code}")
    # Subscribing in on_connect()
means that if we lose the connection
and
    # reconnect then subscriptions
will be renewed.
    client.subscribe("$SYS/#")
    for topic in STATIC.mapping:
        print(f"Subscribing to
{topic}")
        client.subscribe(topic)
```

10. Define a function to call Airflow:

```
def call(url, username,password,
msg):
    headers = {"Content-type":
"application/json", "Accept":
"application/json"}
    try:
        with requests.session() as
s:
            print(url, msg)
            response = s.post(url,
                                data=jso
                                auth=(us
password),
                                headers=
            print("Status Code",
response.status_code,
response.content)
    except Exception as e:
        print("Failed",e)
```

11. The **`on_message()`** function is the core of our system. This function, given the name of the topic, looks into the configuration and, for each analytic, calls Airflow, passing the current value and the parameters in the configuration file:

```
# The callback for when a PUBLISH
message is received from the server.
def on_message(client, userdata,
msg):
    analytics =
STATIC.mapping.get(msg.topic)
    if analytics:
        for analytic in analytics:
            for analytic_name in
analytic:
                url =
STATIC.airflow_endpoint.format(analytic=
                username =
STATIC.airflow_username
                password =
STATIC.airflow_password
                msg = { "conf": {
                    "params":
analytic[analytic_name],
                    "source":
```

```
msg.topic,
                    "value" :
float(msg.payload) }
                }
                #must be async
                print("async to",
url)
                call(url=url,username=us
password=password, msg=msg)
```

12. Then, define a function to connect to the MQTT server:

```
# connect to the server
def connect(host, port):
    mqttc =
mqtt.Client(mqtt.CallbackAPIVersion.VERS
    mqttc.on_connect = on_connect
    mqttc.on_message = on_message
    print(f"Connecting to  {host},
{port}")
    mqttc.connect(host, port, 60)
    mqttc.loop_forever()
```

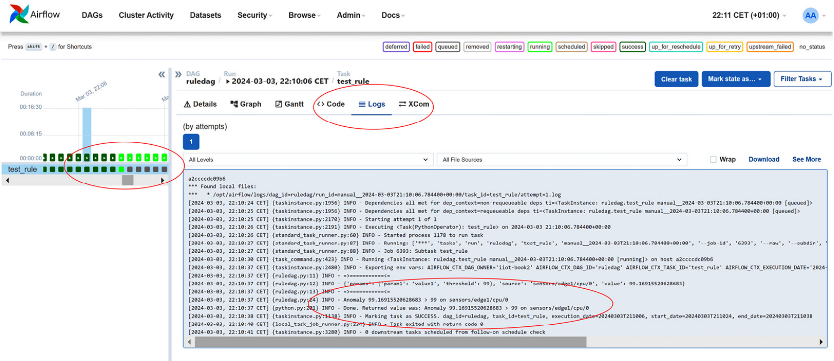
13. Finally, we need a main function to read the configuration file and some **`env`** variables and call the **`connect`** function:

```
if __name__ == "__main__":
    STATIC.airflow_endpoint =
os.getenv('AIRFLOW_ENDPOINT',
"http://localhost:8080/api/v1/dags/
{analytic}/dagRuns")
    STATIC.airflow_username =
    os.getenv('AIRFLOW_USERNAME',
    "airflow")
        STATIC.airflow_password =
    os.getenv('AIRFLOW_PASSWORD',
    "airflow")
        # Read mapping topic analytic
        config_path =
    os.getenv('MAPPING_CONFIG_PATH',
    os.path.join(os.getcwd(), '..',
    'config', 'config.json'))
        with open(config_path) as f:
            STATIC.mapping =
    json.load(f)
        # start
        mqtt_host =
    os.getenv('MQTT_HOST', "mosquitto")
        mqtt_port =
    os.getenv('MQTT_PORT', 1883)
        connect(mqtt_host,mqtt_port)
```

To get a complete solution for the exercises, take a look at the repository at https://github.com/PacktPublishing/Hands-On-Industrial-Internet-of-Things-Second-Edition . In the repo, you can find also the Docker implementation. Indeed, we can start the service with the following:

```
$ docker build .
$ docker compose up
```

Now, we can test our service in action; from the Airflow UI, click on the rule DAG and then on the Logs panel:


Airflow
DAGs
Cluster Activity
Datasets
Security
Browse
Admin
Docs
22:11 CET (+01:00)
AA
Press shift + / for Shortcuts
deferred
failed
queued
removed
restarting
running
scheduled
skipped
success
up_for_reschedule
up_for_retry
upstream_failed
no_status
Duration
00:16:30
00:08:15
00:00:00
Mar 03, 22:08
test_rule
DAG
ruledag
Run
2024-03-03, 22:10:06 CET
Task
test_rule
Clear task
Mark state as...
Filter Tasks
Details
Graph
Gantt
Code
Logs
XCom
(by attempts)
1
All Levels
All File Sources
Wrap
Download
See More

```
a2ccccdc09b6
*** Found local files:
***   * /opt/airflow/logs/dag_id=ruledag/run_id=manual__2024-03-03T21:10:06.784400+00:00/task_id=test_rule/attempt=1.log
[2024-03-03, 22:10:24 CET] {taskinstance.py:1956} INFO - Dependencies all met for dep_context=non-requeueable deps ti=<TaskInstance: ruledag.test_rule manual__2024-03-03T21:10:06.784400+00:00 [queued]>
[2024-03-03, 22:10:25 CET] {taskinstance.py:1956} INFO - Dependencies all met for dep_context=requeueable deps ti=<TaskInstance: ruledag.test_rule manual__2024-03-03T21:10:06.784400+00:00 [queued]>
[2024-03-03, 22:10:25 CET] {taskinstance.py:2170} INFO - Starting attempt 1 of 1
[2024-03-03, 22:10:26 CET] {taskinstance.py:2191} INFO - Executing <Task(PythonOperator): test_rule> on 2024-03-03 21:10:06.784400+00:00
[2024-03-03, 22:10:26 CET] {standard_task_runner.py:60} INFO - Started process 1178 to run task
[2024-03-03, 22:10:27 CET] {standard_task_runner.py:87} INFO - Running: ['***', 'tasks', 'run', 'ruledag', 'test_rule', 'manual__2024-03-03T21:10:06.784400+00:00', '--job-id', '6393', '--raw', '--subdir', '
[2024-03-03, 22:10:27 CET] {standard_task_runner.py:88} INFO - Job 6393: Subtask test_rule
[2024-03-03, 22:10:30 CET] {task_command.py:423} INFO - Running <TaskInstance: ruledag.test_rule manual__2024-03-03T21:10:06.784400+00:00 [running]> on host a2ccccdc09b6
[2024-03-03, 22:10:37 CET] {taskinstance.py:2480} INFO - Exporting env vars: AIRFLOW_CTX_DAG_OWNER='iiot-book2' AIRFLOW_CTX_DAG_ID='ruledag' AIRFLOW_CTX_TASK_ID='test_rule' AIRFLOW_CTX_EXECUTION_DATE='2024-
[2024-03-03, 22:10:37 CET] {ruledag.py:11} INFO - =>===========<=
[2024-03-03, 22:10:37 CET] {ruledag.py:12} INFO - {'params': {'param1': 'value1', 'threshold': 99}, 'source': 'sensors/edge1/cpu/0', 'value': 99.16915520628683}
[2024-03-03, 22:10:37 CET] {ruledag.py:13} INFO - =>===========<=
[2024-03-03, 22:10:37 CET] {ruledag.py:24} INFO - Anomaly 99.16915520628683 > 99 on sensors/edge1/cpu/0
[2024-03-03, 22:10:37 CET] {python.py:201} INFO - Done. Returned value was: Anomaly 99.16915520628683 > 99 on sensors/edge1/cpu/0
[2024-03-03, 22:10:38 CET] {taskinstance.py:1138} INFO - Marking task as SUCCESS. dag_id=ruledag, task_id=test_rule, execution_date=20240303T211006, start_date=20240303T211024, end_date=20240303T211038
[2024-03-03, 22:10:40 CET] {local_task_job_runner.py:234} INFO - Task exited with return code 0
[2024-03-03, 22:10:41 CET] {taskinstance.py:3280} INFO - 0 downstream tasks scheduled from follow-on schedule check
```

Figure 8.18 – A simple rule-based analytic that checks CPU usage

This analytic is very simple, but it has some advantages and disadvantages:

- Advantages :
    - It can be used also to call a complex analytic.
    - It is easy to configure.
    - It can scale up with Docker.

- Disadvantage :
    - It is not so fast; indeed, calling airflow via HTTP is very slow. An alternative solution is to use

`PubSubTopicCreateOperator`, which works very well on GCP.

What's still missing here?

We have discovered the capabilities of Apache Airflow to orchestrate our analytics and develop extensions for our data source. To make Apache Airflow a platform for IIoT, however, we need a connector to our asset registry and a simple task able to run analytics for each asset we want to monitor. It is very easy to extend our plugin for InfluxDB to Neo4j, using the Neo4j driver for Python. This is described in more detail in the following section. To scale the execution of our analytics across our fleet according to our asset list, we can build our DAG dynamically.

Building an asset registry to store asset information

In *Chapter 7* , we highlighted the importance of the asset registry and its role within the context of the IIoT. The asset registry can maintain a lot of information, including the class of the model (the asset model) of our equipment (the asset instance), the ID of the equipment and its properties, the components of the equipment, the measures we must monitor, the thresholds and functional parameters of our measures, and the hierarchy of our fleet. A typical hierarchy of our assets might look as follows:

```
Asset Instance ID:Asset Class ID ->
Asset Instance Child ID:Asset Class ID
```

In the following example, **`ACME: Enterprise`** manages **`SydneyPlant`** . This, in turn, hosts a subsystem, which includes **`CT001`** :

```
ACME:Enterprise -> SydneyPlant:Plant -
> Train1:Segment -> CT001:Pump
```

`CT001` is a pump, which we can use to connect several sensors to monitor – for instance, the temperature:

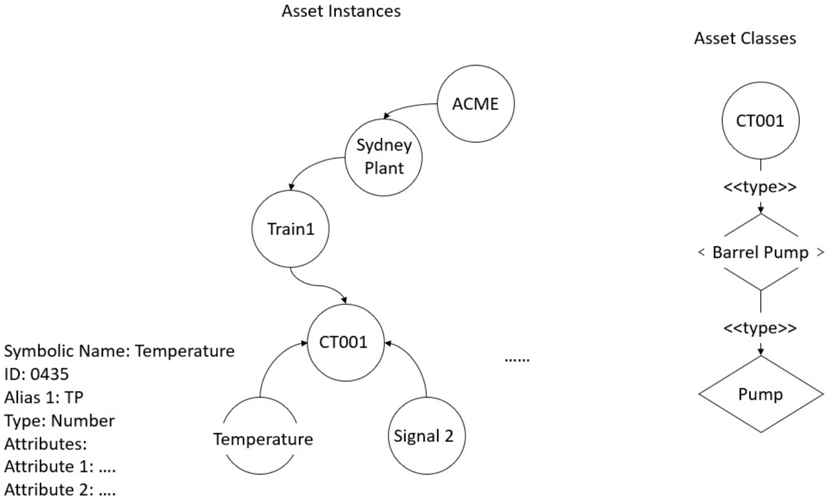

Asset Instances
Asset Classes
ACME
Sydney Plant
Train1
CT001
Symbolic Name: Temperature
ID: 0435
Alias 1: TP
Type: Number
Attributes:
Attribute 1: ....
Attribute 2: ....
Temperature
Signal 2
......
CT001
<<type>>
< Barrel Pump >
<<type>>
Pump

Figure 8.19 – A simple asset model

A consumer (e.g., analytic, human, or hardware device) of an asset registry could simply ask, “ *Give me all the measures of CT001, or, give me all the temperatures of the assets belonging to Train1.* ” An asset registry could manage thousands of assets and millions of measures, so we need an efficient mechanism to store and retrieve this information. It is very easy to understand why a graph database is better than a relational database. A graph database is a database that stores information about nodes and the relationships among them. An efficient graph database is Neo4j (https://neo4j.com/).

> ***Tip***

> *IS-95 is a standard for building asset models. An asset registry should also support different versions of the same asset to allow backward or forward compatibility with different tools.*

Building an asset model with Neo4j

The easiest way to run Neo4j is by using the official Docker image:

```
$ docker run -p 7474:7474 -p 7687:7687
--network iiot-book-2-net neo4j
```

Alternatively, you can use the Docker Compose file. Make a directory, `neo4j` , with the following `docker-compose.yml` :

```
version: "3"
services:
  neo4j:
    container_name: neo4j
    image: neo4j:4.4.31-community
    ports:
      - "7474:7474"
      - "7687:7687"
    environment:
    - NEO4J_AUTH=neo4j/admin
    -
NEO4J_ACCEPT_LICENSE_AGREEMENT=yes
    - NEO4J_EDITION=docker_compose
    networks:
    - iiot-book-2-net
networks:
    iiot-book-2-net:
      external: true
```

Then, start the service with the following:

```
$ docker compose up
```

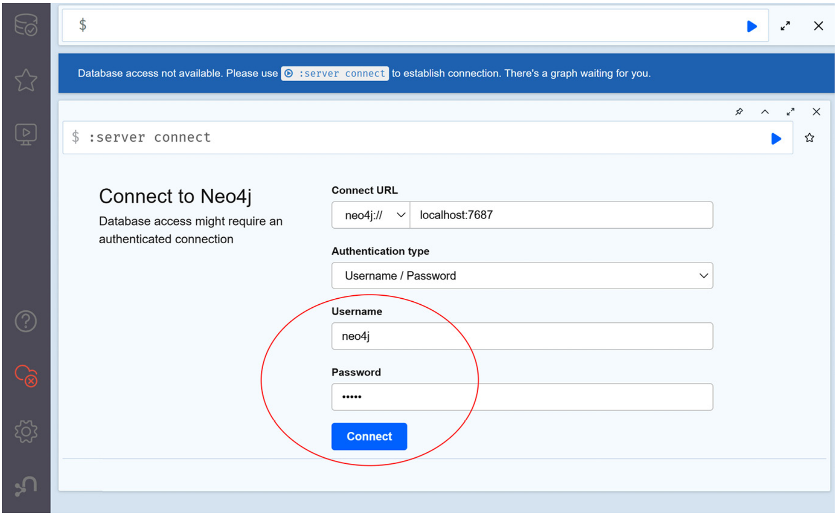
To get a complete solution for the exercises, take a look at the repository at https://github.com/PacktPublishing/Hands-On-Industrial-Internet-of-Things-Second-Edition .

The iiot-book-2-net network

> *In this section, we assume that a network with the name* **`iiot-book-2-net`** *has been already created in the host machine. This network was created in the first section, but if there is an error, run the following command from the command console –* **`docker network create -d bridge iiot-book-2-net`**.

Neo4j supports a powerful query and a create, read, update, and delete (CRUD) language called Cypher , and it also supports the REST API interface, Java, Python, C#, as well as JavaScript drivers to work with Cypher.

When the service is up and running, we can access the Neo4j UI; we can open the browser at http://localhost:7474/browser/ . On the splash screen, connect to **`localhost:7687`** with the username **`neo4j`** and the password **`admin`** . The following screenshot shows the UI of Neo4j:



```
$
```

Database access not available. Please use `:server connect` to establish connection. There's a graph waiting for you.

```
$ :server connect
```

Connect to Neo4j

Database access might require an authenticated connection

Connect URL: neo4j:// localhost:7687

Authentication type: Username / Password

Username: neo4j

Password: •••••

Figure 8.20 – The Neo4j UI

Now, we can build our asset model, as depicted in the previous section with Cypher. The syntax is very intuitive.

To create an asset, follow these steps:

```
CREATE ASSETNAME:TYPE {ATTRIBUTES}
```

To create a relationship among assets, execute the following:

```
CREATE ASSETNAME -[RELATIONSHIP]->
ASSETNAME
To search for:
MATCH(n) RETURN n
```

Let's take a look at the further steps:

1. Create the asset : On the Neo4j UI, we can write the following commands:

```
CREATE (CT001:Pump {name:'CT001',
alias:'Pump-SN-993416776',
model:'standard'}),
       (Train1:Section
{name:'Production Train 1'}),
       (SydneyPlant:Plant
{name:'Plant of ACME in Sydney'}),
       (ACME:Company {name:'ACME
International'}),
```

2. Link the assets : Create a relationship among the assets:

```
      (CT001)-[:BELONGING_OF {roles:
['Part of']}]->(Train1),
      (Train1)-[:BELONGING_OF
{roles:['Part of']}]->(SydneyPlant),
      (SydneyPlant)-[:BELONGING_OF
{roles:['Managed by']}]->(ACME),
```

3. Create the final measures : Create the measures (tags) and associate them with the **`CT001:Pump`** asset:

```
      (TEMP01:Measure
{name:'CT001.TEMPERATURE01',
alias:'TEMP01', type:'TEMPERATURE',
uom:'DEG'}),
        (FLOW01:Measure
{name:'CT001.FLOW01',
alias:'FLOW01',type:'FLOW',
uom:'sm3/sec'}),
```

4. Add another measure, **`CPU`**, with some additional parameters – **`topic`**, **`threshold`**, and **`analytic`**; we will reuse them later. The following is the code:

```
        (CPU:Measure
{name:'CT001.CPU',
alias:'CPU',type:'cpu', uom:'perc',
topic:'sensors/edge1/
cpu/0',analytic:'ruledag',
threshold:'99'}),
```

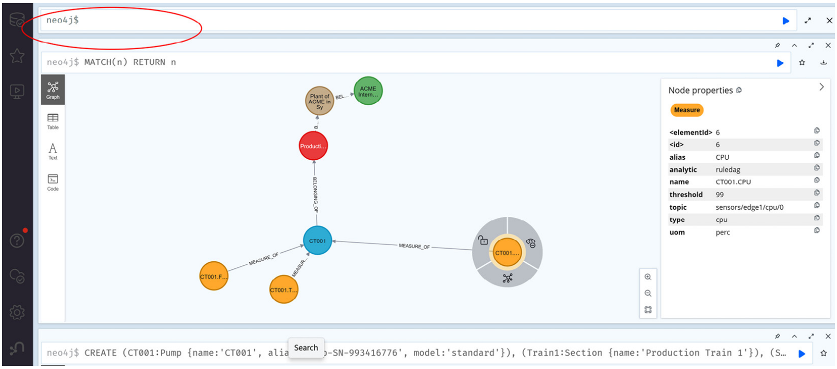
5. Then, link it to the asset:

```
        (TEMP01)-[:MEASURE_OF]-
>(CT001),
        (FLOW01)-[:MEASURE_OF]-
>(CT001),
        (CPU)-[:MEASURE_OF]->(CT001)
```

6. Finally, on the UI, run the following command:

```
$ MATCH (n) RETURN n
```

The result of this is as follows:


Graph
Table
Text
Code
Plant of ACME in Sy
ACME Intern...
BEL...
Producti...
BELONGING_OF
CT001
MEASURE_OF
CT001.F...
CT001.T...
MEASUR...
CT001....
Search

```
neo4j$
```

```
neo4j$ MATCH(n) RETURN n
```

Node properties

Measure

<elementId>	6
<id>	6
alias	CPU
analytic	ruledag
name	CT001.CPU
threshold	99
topic	sensors/edge1/cpu/0
type	cpu
uom	perc

```
neo4j$ CREATE (CT001:Pump {name:'CT001', alia -SN-993416776', model:'standard'}), (Train1:Section {name:'Production Train 1'}), (S…
```

Figure 8.21 – The output shown on Neo4j

Working with Neo4j

Neo4j has a very flexible language, and we can work with Cypher using Python, JavaScript, C#, or the Java driver. To work with Python, we need to install the driver:

```
$ pip install neo4j
```

We are now ready to develop our code. Build the `ask_for_measure.py` file with the following code:

1. Import the packages:

```
from neo4j import GraphDatabase
```

2. Define a class name with three methods – `connect`, `close`, and `run` query:

```
class Neo4JClient:
    def __init__(self, uri, user,
password):
        self.driver =
GraphDatabase.driver(uri,
auth=(user, password))
    def close(self):
        self.driver.close()
    def query(self, query):
        with self.driver.session()
as session:
            ret = session.run(query)
            return ret.data()
```

3. Now, we can write our main method to ask for every topic name in the measures:

```
if __name__ == "__main__":
    client = Neo4JClient("bolt://
localhost:7687", "neo4j", "admin")
    ret = client.query('MATCH
(M:Measure {topic:"sensors/edge1/
cpu/0"})-[BELONGING_OF]->(A) RETURN
M.name, M.analytic, A.name')
    print(ret)
    ret = client.query('MATCH
(M:Measure) RETURN M.name, M.topic')
    print(ret)
```

4. We can then simply run the following command:

```
$ python ask_for_measure.py
```

This will give us the following result:

```
[{'M.name': 'CT001.CPU', 'M.analytic':
'ruledag', 'A.name': 'CT001'}]
[{'M.name': 'CT001.TEMPERATURE01',
'M.topic': None}, {'M.name':
'CT001.FLOW01', 'M.topic': None},
{'M.name': 'CT001.CPU', 'M.topic':
'sensors/edge1/cpu/0'}]
```

The first query gets measures and the related asset connected to the measures. The second query looks for all measures and extracts the list of topics.

Integrating Neo4j into a data flow

Now, we are going to integrate the asset registry into our data flow. It is good practice to use the asset registry to keep the configuration of the data flow. An asset registry is a database where we can store static information about the assets fleet.

In the previous exercise, we configured a measurement called *CPU* and linked it to the *CT001* pump. We have also defined some properties to link the CPU measurement and the analytic to be processed (in this case, just one, but there could be many). The idea is to replace the static configuration of the `airflowmqtt` connector with a more dynamic and centralized configuration.

Open the `main.py` file, located in `src` of the `airflowmqtt` project, and add the previous Neo4JClient class and a new class:

```
class Neo4JConfigParser(Neo4JClient):
    def get_config(self):
        ret = {}
        measures = self.query('MATCH
(M:Measure)-[BELONGING_OF]->(A) RETURN
M.name, M.topic, M.threshold,
M.analytic, A.name')
        for i in measures:
            topic = i["M.topic"]
            if topic is not None:
                ret[topic] =[
                    {i["M.analytic"]:
{"asset": i["M.name"],
"threshold":i["M.threshold"]}}
                    ]
        return ret
```

As shown previously, the current query extracts all the measures, and if a measure contains a **`topic`** property, it adds to the configuration file a new item with the analytic to call and some parameters.

Then, replace the mapping code of the service in the main method with the following code:

```
    # Read mapping topic analytic
    neo4j_url =
os.getenv('NEO4J_ENDPOINT', None)
    neo4j_username =
os.getenv('NEO4J_USERNAME', "neo4j")
    neo4j_password =
os.getenv('NEO4J_PASSWORD', "admin")
    if neo4j_url is not None:
        config =
Neo4JConfigParser(neo4j_url,
neo4j_username, neo4j_password)
        STATIC.mapping =
config.get_config()
        config.close
    else:
        config_path =
os.getenv('MAPPING_CONFIG_PATH',
os.path.join(os.getcwd(), '..',
'config', 'config.json'))
        with open(config_path) as f:
            STATIC.mapping =
  json.load(f)
```

Start the new service using Docker Compose or the command line:

```
$ set NEO4J_ENDPOINT=bolt://
localhost:7687
$ python main.py
```

In this section, we learned the importance of asset registry. The asset registry is the single point of truth, where the mapping between the assets, the measurements, the analytics to call, and the configuration parameters can be found. In industrial sites, it is typical to synchronize the asset registry with enterprise resource planning (ERP) or a manufacturing execution system (MES) as a single source of truth.

The asset registry should also contain information about the machine we are monitoring, such as the model, when it was installed, and the production environment. In fact, the analytic could make different decisions based on this information. For example, it may accept lower performance if the machine was installed many years ago. Moreover, Airflow supports the Neo4J operator, so we can use **`apache-airflow-providers-neo4j`** to extract asset information inside the DAG on Airflow.

> ### *Overall Equipment Effectiveness (OEE)*

> *OEE = availability x performance x quality; OEE is the measure of the total effectiveness of a system.* ***Availability*** *is a fraction of the allocated time that a facility is actually available.* ***Performance*** *represents the speed at which a system works as a fraction of the design speed. The* ***quality*** *rate indicates the percentage of units in specification compared to all those produced.*

Congratulations! We have just developed our first IIoT architecture. We learned how to connect the edge device via an IoT protocol (using MQTT). We visualized the data. Finally, we developed our first data flow.

We will review the same concepts in the next chapters and implement the same flow in AWS or

Azure.

Pros and cons of the proposed platform

The platform we have proposed in this chapter has highlighted several important concepts about IIoT and data processing. Let's look at some of its advantages and disadvantages.

Here are some advantages:

- The data is stored locally or in an on-premises cloud that is vendor-agnostic
- We can use a federated architecture to grant the right ownership of the intellectual property
- We can plug in new applications, avoiding the limitation of a specific technology
- Reliability can be improved by configuring multiple nodes of Docker instances

Here are some disadvantages:

- Edge management is completely missing here. This includes features, such as patching, security, or certificate renewal, that are normally supported by a vendor.

- The current solution doesn't include an MES and ERP. A vendor supports these connectors natively.
- Only one protocol, MQTT, is supported.

The first three issues can be improved and managed easily, but the final topic is very important for an IIoT platform. In a typical IoT platform, we have a few models and thousands of machines, with a few measures that we have to monitor. The ratio from models to assets to measures is models-assets-measures – 10-100.000-10. In the IIoT, however, we have dozens of models, fewer machines, and hundreds of measures per asset to monitor, so the ratio from models to assets to measures is 100:10,000:100. This means that we need an efficient mechanism to scale and distribute the analytics across the fleet, using the asset model. In other words, a real platform should support both a catalog of analytics and the capability to attach an analytic to a specific asset, using the definition or the attributes of the model. For instance, we should avoid monitoring the corrosion of an onshore wind turbine against an offshore wind turbine, or deploying one analytic to a two-stage elevator against a single-stage elevator.

Other technologies

In the previous sections, we have implemented an architecture based on the most common technologies. There are other technologies that can replace the ones we propose. In this section, we will learn about some of them.

Other open source technologies for analytics

Airflow is, without a doubt, a very valid platform for building analytics. We can also implement our analytics on other data platforms. Here is a short list of them:

- Apache Flink is a data-stream processor with great support for streaming analytics. The major advantage of Flink is that it has stateful support.

- Dagster is a valid alternative to Airflow.
- Kedro simplifies the data pipeline.
- Kubeflow is based on Kubernetes and Docker to orchestrate data flow among dockers.
- Apache NiFi is a powerful Extract, Transform, Load (ETL) orchestrator system.
- Luigi is a very thin orchestrator analytic platform.
- Apache Spark is the most popular framework for big data and stream processing. Apache Spark can run on a Yet Another Resource Negotiator (YARN) cluster, a native cluster, or Mesos. Several IoT cloud platforms have support for Spark, including AWS and Azure.

> ### *Big data versus IoT analytics*

> *IoT platforms are often confused with big data platforms. The IoT flow is generally a continuous flow that does not require massive analyses but, rather, timely and continuous analyses. Many companies confuse big data and IoT; to date, big data platforms in the IoT sector are only used for one-shot or massive analyses. Big data is a valid ally of the IoT, but it is not the last solution.*

Other open source platforms

Other open source platforms are available as valid alternatives to the previously proposed technologies. Here is a short list of the most interesting platforms or technologies:

- RabbitMQ : RabbitMQ is a queue messaging system based on Advanced Message Queuing Protocol (AMQP). It has great support for plugins and routing. RabbitMQ is a valid alternative to Apache Kafka. On the IoT, it is very common to use AMQP for the data protocol.
- Apache ActiveMQ : A valid alternative to RabbitMQ is Apache ActiveMQ, which supports multiple protocols.
- Redis : Redis is an in-memory NoSQL database that is normally used for caching.
- Elasticsearch : Elasticsearch is a very popular indexing engine based on Apache Lucene. Elasticsearch can be used to store logs and information after the data-processing stage.
- Kibana : Kibana is based on Elasticsearch and provides a very flexible way to build dashboards and reports.

Other IIoT data beyond the time series

Both the IIoT and the IoT are, by default, linked to signal acquisition and data processing. In less standard cases, however, we may have to process unstructured data or raw data, such as images, log files, or binary files, if, for example, we were processing geological drilling images. Normally, this information is stored in a data lake and processed at a later stage through big data analytics.

Summary

In this chapter, we learned how to develop an IIoT platform from scratch. The purpose of this was to highlight the issues we might encounter during the platform's development.

We isolated the edge from the centralized data database by studying a widespread protocol such as MQTT.

We studied the differences between the cold path and the hot path (continuous versus batch). We did some exercises with the asset registry to understand the importance of having a single point to keep all the relationships between the various assets and the related measures. In the next chapter, we will explore the most common IoT and IIoT platforms.

9

Building an AWS Industrial IoT Solution

In the previous chapter, we implemented a custom industrial IoT (IIoT) platform. We looked at some important topics, including storage, time series, microservices, protocols, data stream processing, and batch processing. This chapter explores the solutions proposed by Amazon Web Services (AWS) and the capabilities of the AWS IoT platform.

In this chapter, we will discover some of the most popular cloud solutions that apply to IIoT. We will cover the following topics:

- Registering with AWS
- The AWS architecture
- AWS IoT core

- Greengrass and SiteWise

We will continue to explore the AWS IoT architecture in the next chapter (*Chapter 10*), where we will implement an end-to-end data flow using the following technologies:

- Storing data in Timestream or S3
- AWS IoT Analytics
- Grafana

Technical requirements

To follow along with this chapter, the following prerequisites should be downloaded and installed:

- Git: https://git-scm.com/downloads
- Node.js 20+: https://nodejs.org
- Python 3.9+: https://www.python.org/downloads
- Docker: https://www.docker.com/products/docker-desktop/

The code for this chapter is available in the official repository at https://github.com/PacktPublishing/Hands-On-Industrial-Internet-of-Things-Second-

Edition .

Why do we use commercial IIoT platforms?

Why should we use commercial IIoT platforms? Why should we use the cloud? The answer is clear—commercial IIoT platforms are cost-effective and mature platforms that provide excellent levels of productivity, fast development, deployment automation (DevOps), reliability, availability, and auto scalability. However, there are some reasons why we might want to avoid them—for instance, if we need a high level of data protection and control of exports, if we need high flexibility for custom integration and proprietary protocols, or if we would like to avoid dependencies to a specific vendor.

All these issues can be mitigated or resolved, but the choice to adopt a particular platform depends on several factors. The following figure shows a simple decision-making diagram:

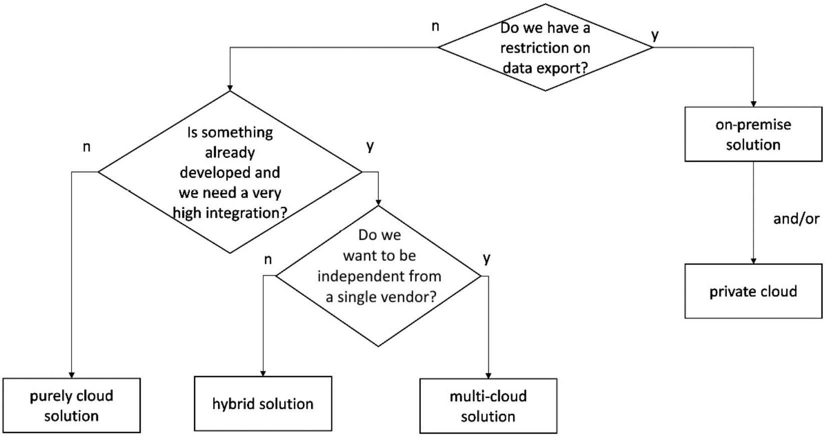

n
Do we have a restriction on data export?
y
Is something already developed and we need a very high integration?
n
y
on-premise solution
and/or
Do we want to be independent from a single vendor?
n
y
private cloud
purely cloud solution
hybrid solution
multi-cloud solution

Figure 9.1 – Workflow to choose between cloud versus on-premises solutions

First, we need to check whether we have any restrictions on the export of data. If we don't, we might opt for a cloud solution. If we can build everything from scratch, we can use a vendor cloud solution. Otherwise, we can choose between a hybrid solution or a multi-cloud solution.

Generally speaking, if we do not want to adopt a single strategy, we can keep the standard business on-premises and migrate to a cloud platform later.

This means we can use the best functionalities provided by each cloud vendor by applying a multi-cloud solution that uses inter-cloud communications.

AWS architecture

In previous chapters, we designed an end-to-end standard architecture from the sensor to the operator. In this chapter, we will discover that AWS has a lot of similarities with the architecture proposed in *Chapter 8* ; this cloud architecture is strongly based on microservices, queues, and common protocols, including HTTPS, MQTT, and REST APIs. From the point of view of infrastructure, AWS supports Kubernetes, Hadoop, Docker, and serverless platforms. From an IoT perspective, AWS has built its own solution: AWS IoT .

AWS IoT

AWS IoT is the IoT platform of AWS. The key components of AWS IoT are the following:

- AWS Greengrass
- AWS SiteWise
- AWS IoT Core
- AWS IoT Analytics

Other components of AWS that are not specific to the IoT are listed as follows:

- Amazon Managed Grafana
- Amazon Managed Service for Prometheus
- Amazon SageMaker
- Amazon Timestream
- AWS Lambda
- Amazon S3

The following diagram should be a full architecture for the IIoT using AWS:

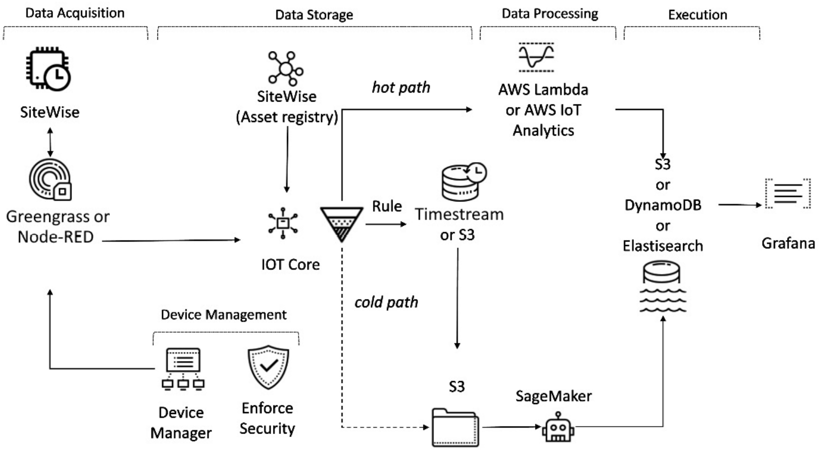

Data Acquisition
Data Storage
Data Processing
Execution
SiteWise
Greengrass or Node-RED
SiteWise (Asset registry)
IOT Core
hot path
Rule
AWS Lambda or AWS IoT Analytics
Timestream or S3
S3 or DynamoDB or Elastisearch
Grafana
cold path
Device Management
Device Manager
Enforce Security
S3
SageMaker

Figure 9.2 – Architecture reference

IoT Core is the hub to which the devices send data using several protocols. In our exercise, this was MQTT. The data can be stored in DynamoDB for time-series data, or S3 for object data. AWS allows us to perform this storing action by enabling simple rules from IoT Core. This is further explained in *Chapter 10* 's AWS IOT analytics section. In parallel, we can process data using AWS Lambda, a serverless platform, or using IoT Analytics. Results can either be stored on S3 or Elasticsearch, or directly connected to Grafana for fast visualizing. AWS also allows you to export data from

Timestream to S3 with a few clicks. We can use AWS IoT Analytics, and SageMaker for batch processing. Finally, AWS allows us to carry out actions on the edge by deploying AWS Lambda on-premises using Greengrass.

To discover these features, we will develop a simple exercise based on IoT Core, AWS Lambda, MQTT, the AWS Edge SDK, IoT Analytics, and Grafana. In this chapter, we will only cover the first part by implementing the edge-to-cloud flow, while in *Chapter 10* , we will complete our exercise by implementing the operational data flow, from storage to analytics to visualization:

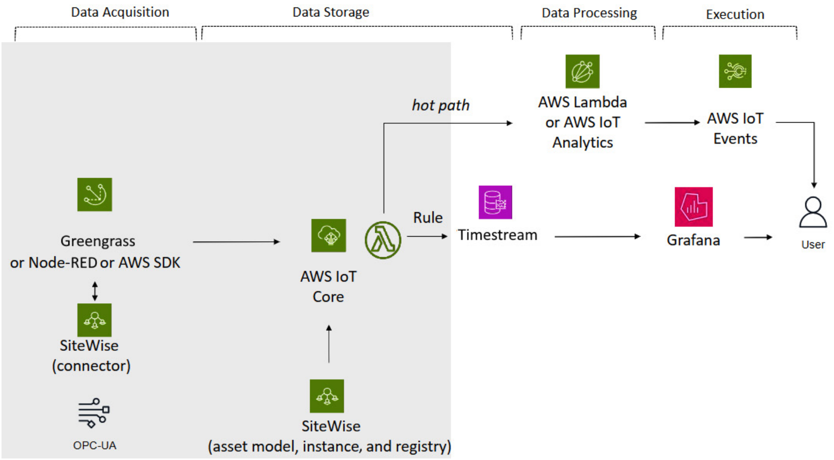

Data Acquisition
Data Storage
Data Processing
Execution
hot path
AWS Lambda or AWS IoT Analytics
AWS IoT Events
Greengrass or Node-RED or AWS SDK
AWS IoT Core
Rule
Timestream
Grafana
User
SiteWise (connector)
OPC-UA
SiteWise (asset model, instance, and registry)

Figure 9.3 – Architecture of our exercise

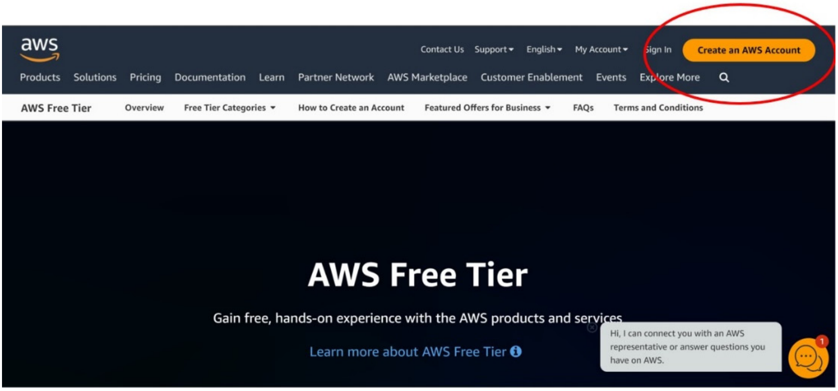
To make a start with AWS IoT, we need a valid AWS account.

Registering with AWS

To work with AWS, we need to register for a Free Tier account.

Open the browser and go to https://aws.amazon.com/free , then create a free account as follows:

1. Create a free account in AWS by clicking on the button highlighted in the following figure:

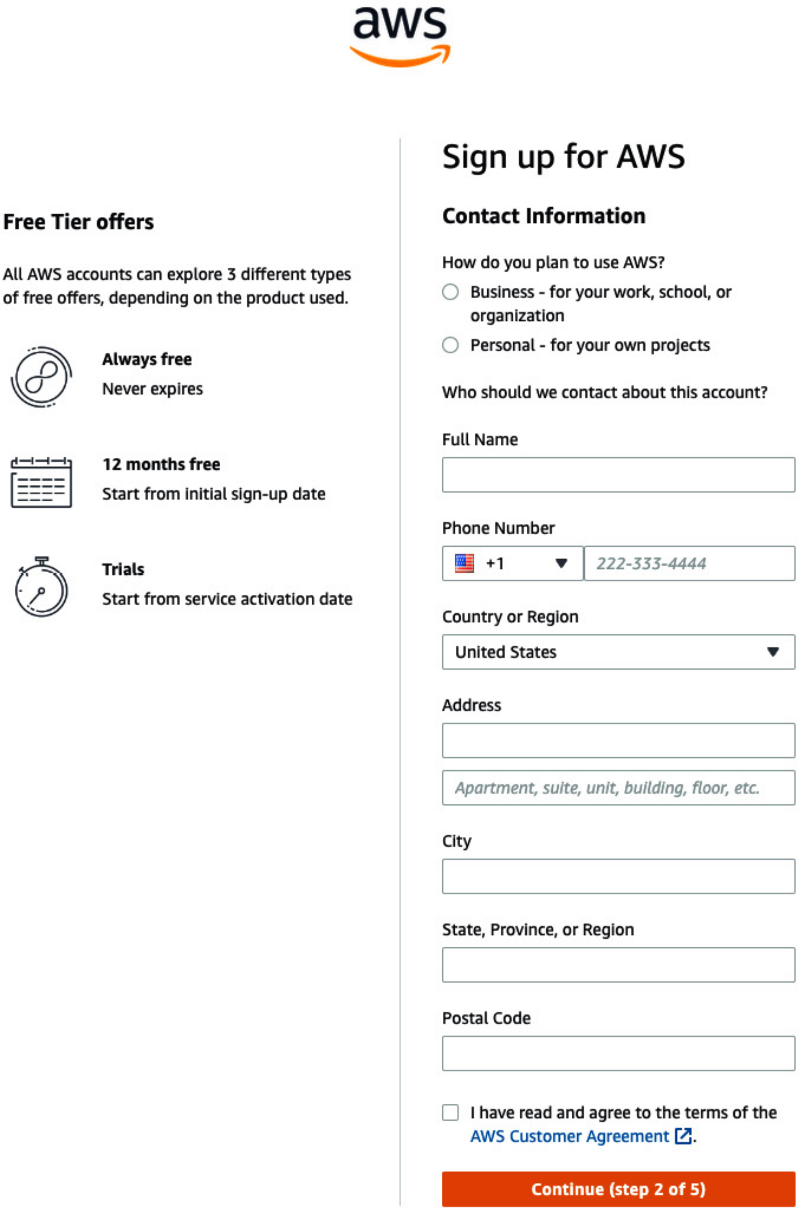

aws
Contact Us
Support
English
My Account
Sign In
Create an AWS Account
Products
Solutions
Pricing
Documentation
Learn
Partner Network
AWS Marketplace
Customer Enablement
Events
Explore More
AWS Free Tier
Overview
Free Tier Categories
How to Create an Account
Featured Offers for Business
FAQs
Terms and Conditions
AWS Free Tier
Learn more about AWS Free Tier

Gain free, hands-on experience with the AWS products and services

Hi, I can connect you with an AWS representative or answer questions you have on AWS.

Figure 9.4 – Creating a free AWS account

1. You will need to provide an email address and an account name and verify the email address in your email inbox.
2. Create your root user and password; please do not forget these, as we will use them during the next steps.
3. Add contact information and choose a plan (Personal is enough for this exercise):



Free Tier offers

All AWS accounts can explore 3 different types of free offers, depending on the product used.

Always free
Never expires

12 months free
Start from initial sign-up date

Trials
Start from service activation date

Sign up for AWS

Contact Information

How do you plan to use AWS?

○ Business - for your work, school, or organization
○ Personal - for your own projects

Who should we contact about this account?

Full Name

Phone Number
+1 ▼ 222-333-4444

Country or Region
United States ▼

Address

Apartment, suite, unit, building, floor, etc.

City

State, Province, or Region

Postal Code

☐ I have read and agree to the terms of the AWS Customer Agreement.

Figure 9.5 – Creating a free AWS account part 2

1. Add a payment method; AWS will test this by taking a payment of 1 euro. AWS will not charge you if you are compliant with their Free

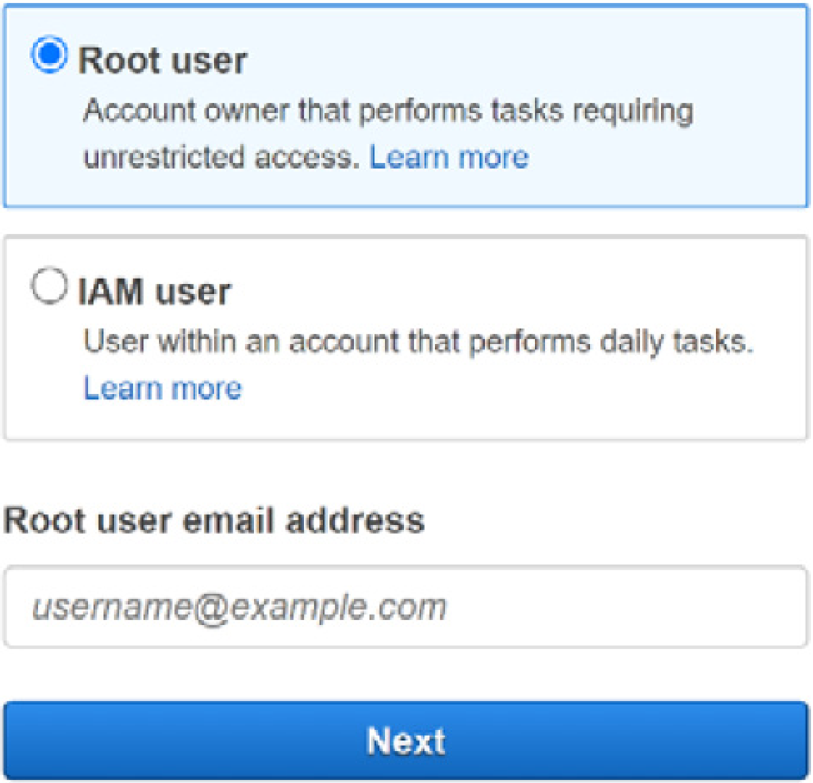
Tier license.

2. Choose how you want to confirm your identity.
3. Select a support plan; Basic is enough for our exercise.

AWS account validation

> *AWS may require a few more steps and 24 hours for final validation.*

1. When the account is ready, sign in to the AWS Management Console at https://aws.amazon.com and provide your root email username and password:



● **Root user**

Account owner that performs tasks requiring unrestricted access. Learn more

○ **IAM user**

User within an account that performs daily tasks. Learn more

Root user email address

username@example.com

Next

Figure 9.6 – Logging in to your AWS account

1. Choose your AWS default location:

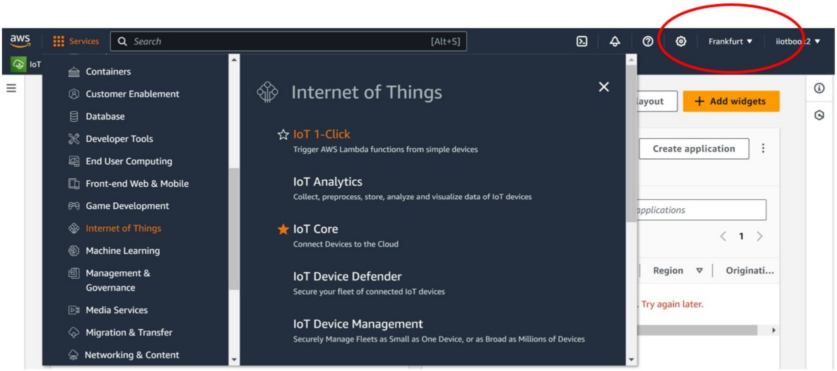

aws
Services
Search
[Alt+S]
Frankfurt ▼
iiotboo 2 ▼
IoT
Containers
Customer Enablement
Database
Developer Tools
End User Computing
Front-end Web & Mobile
Game Development
Internet of Things
Machine Learning
Management & Governance
Media Services
Migration & Transfer
Networking & Content
Internet of Things
IoT 1-Click
Trigger AWS Lambda functions from simple devices
IoT Analytics
Collect, preprocess, store, analyze and visualize data of IoT devices
IoT Core
Connect Devices to the Cloud
IoT Device Defender
Secure your fleet of connected IoT devices
IoT Device Management
Securely Manage Fleets as Small as One Device, or as Broad as Millions of Devices
ayout
+ Add widgets
Create application
applications
1
Region
Originati...
Try again later.

Figure 9.7 – Choosing the AWS location

In our case, we activated Europe Central 1 (Frankfurt).

> ### *Why zones?*

> *Although all cloud services can work worldwide, due to local laws and policies on data retention (and personal data), it may be necessary to ensure that the services are instantiated and consumed in the region where they were generated. IIoT data is not personal data but can provide very sensitive business information. Check the contract with the final customer and any export control restrictions.*

1. It is a good practice to activate multifactor authentication (MFA); If you don't have access to a mobile device or a hardware device, you cannot enable MFA.

   From the menu on the right, choose Security credentials :

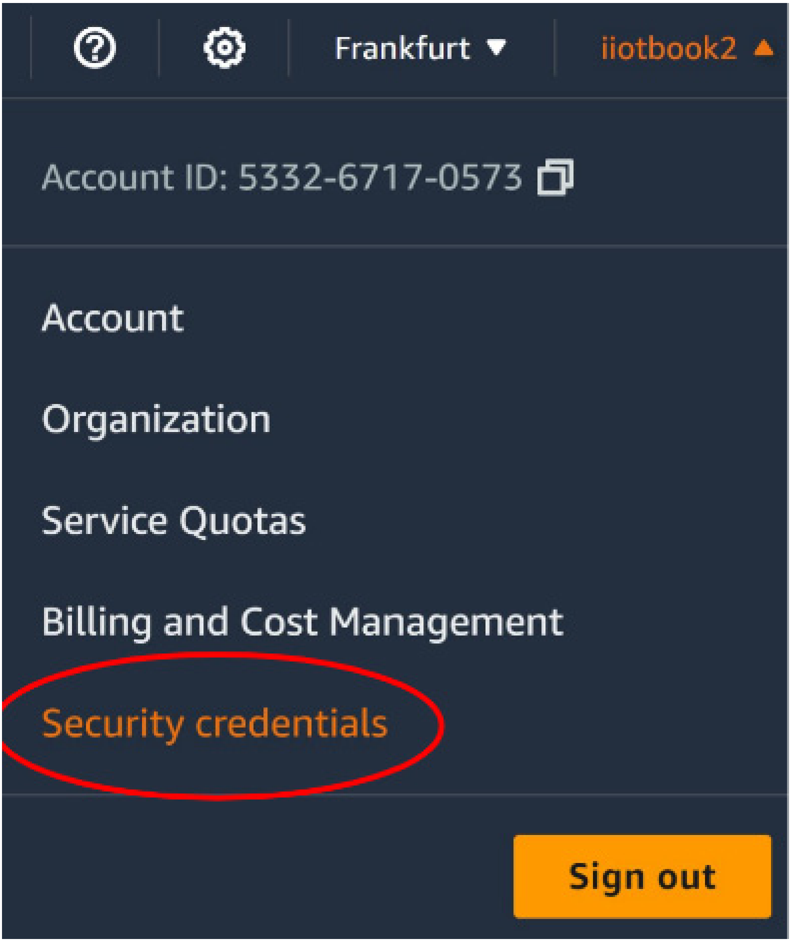

Frankfurt ▼
iiotbook2 ▲
Account ID: 5332-6717-0573
Account
Organization
Service Quotas
Billing and Cost Management
Security credentials
Sign out

Figure 9.8 – MFA

1. Then, follow these steps:

    I. Click on Assign MFA .

    II. Scan the QR code with your preferred authenticator. Phone app authenticators work very well.

    III. Provide two key codes.

2. It is considered a security best practice to not use your root account for everyday tasks.

   Connect to the AWS Management Console at **`https://<your zone>.aws.amazon.com/console`** and search for the Identity Access Manager (IAM) service:

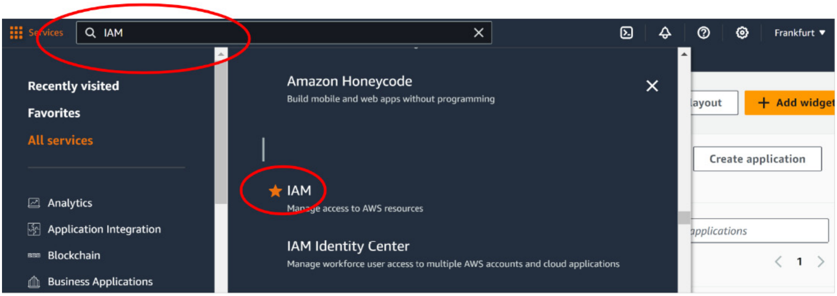

Services
IAM
Frankfurt
Recently visited
Favorites
All services
Analytics
Application Integration
Blockchain
Business Applications
Amazon Honeycode
Build mobile and web apps without programming
IAM
Manage access to AWS resources
IAM Identity Center
Manage workforce user access to multiple AWS accounts and cloud applications
ayout
Add widget
Create application
applications
1

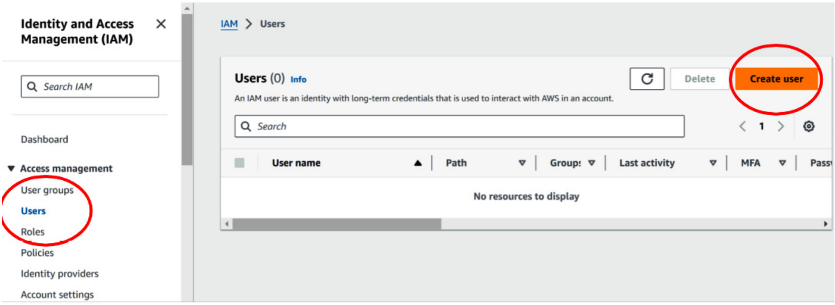
Figure 9.9 – Creating a new user with IAM

> ***Tip***

> *We will repeat this step several times in this chapter and the next; therefore, we suggest you save the services that we will use in your favorites to find them more quickly.*

Alternatively, you can simply open the browser to https://console.aws.amazon.com/iam/ .

1. In the Users tab, select Create user :

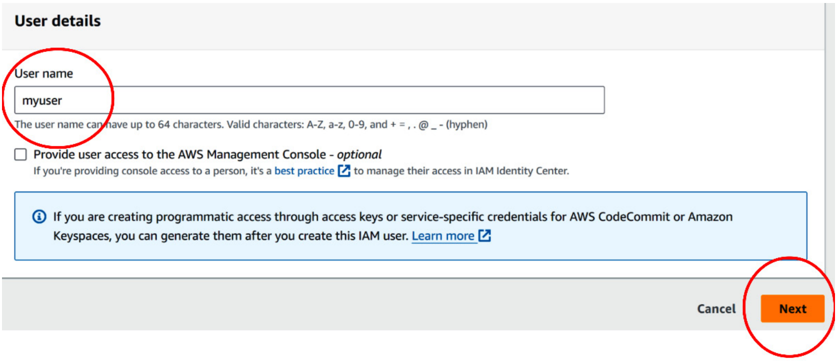

Identity and Access Management (IAM)
Search IAM
Dashboard
Access management
User groups
Users
Roles
Policies
Identity providers
Account settings
IAM > Users
Users (0) Info
Delete
Create user
Search
1
User name
Path
Groups
Last activity
MFA
Pass

An IAM user is an identity with long-term credentials that is used to interact with AWS in an account.

No resources to display

Figure 9.10 – Creating a user

1. Then, enter a username:


User details
Cancel
Next

User name: myuser

The user name can have up to 64 characters. Valid characters: A-Z, a-z, 0-9, and + = , . @ _ - (hyphen)

☐ Provide user access to the AWS Management Console - *optional*

If you're providing console access to a person, it's a best practice to manage their access in IAM Identity Center.

If you are creating programmatic access through access keys or service-specific credentials for AWS CodeCommit or Amazon Keyspaces, you can generate them after you create this IAM user. Learn more

Figure 9.11 – Defining username

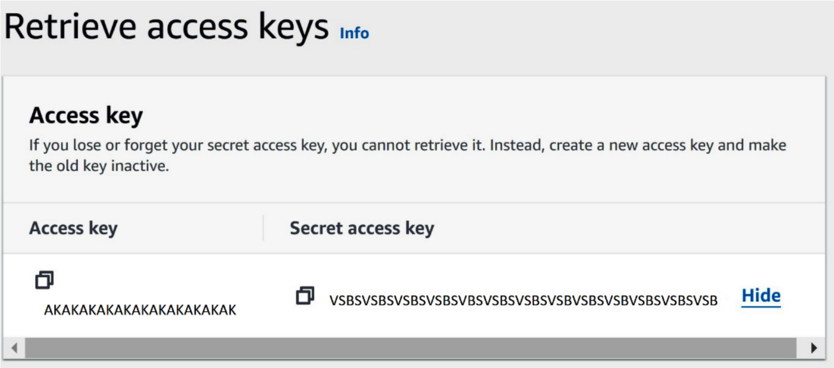
1. To provide administrative permissions, click on Attach existing policies directly and enable Administrator Access .

For our exercise, we need the following:

- `AmazonEC2ReadOnlyAccess`
- `AmazonS3FullAccess`
- `AWSIoTAnalyticsFullAccess`

- `AWSIoTDataAccess`
- `AWSIoTFullAccess`
- `AWSGreengrassFullAccess`

8. Then, click on the Next: Review button and then the Create user button.
9. Finally, save the access key in a protected folder, as we will reuse it later:



Retrieve access keys Info

Access key

If you lose or forget your secret access key, you cannot retrieve it. Instead, create a new access key and make the old key inactive.

Access key	Secret access key
AKAKAKAKAKAKAKAKAKAKAK	VSBSVSBSVSBSVSBSVBSVSBSVSBSVSBVSBSVSBVSBSVSBSVSB

Figure 9.12 – Saving the access key to reuse it later (these access keys are dummy access keys)

In this section, we saw how to work with AWS IAM to create users; we will review IAM several times throughout this chapter. Next, let's learn about another important concept.

Installing the AWS command-line interface client

In this section, we will use the AWS command-line interface (CLI) client (AWS CLI 2). To install it, follow the instructions described at https://docs.aws.amazon.com/it_it/cli/latest/userguide/getting-started-install.html .

On Windows, from the CLI, enter the following command:

```
$ msiexec.exe /i https://
awscli.amazonaws.com/AWSCLIV2.msi
```

On Linux, from the CLI, enter the following command:

```
$ curl "https://awscli.amazonaws.com/
awscli-exe-linux-x86_64.zip" -o
"awscliv2.zip"
$ unzip awscliv2.zip
$ sudo ./aws/install
```

We can then configure AWS with the following command:

```
$ aws configure
```

We can use the access key and secret access key generated before, then define the region name (for example, `eu-central-1`).

To test it, run the following from the CLI:

```
$ aws s3api list-buckets
```

The output should be similar to this:

```
{
    "Buckets": [],
    ...
}
```

In this section, we have seen how to work with the AWS CLI and have completed the settings of our prerequisites. In the next section, we will begin to create our IIoT architecture with AWS.

IoT Core

IoT Core is a basic service to manage devices and receive data from an edge device. To enable IoT Core, we need to open the web console and click on IoT Core , as shown in the following screenshot:

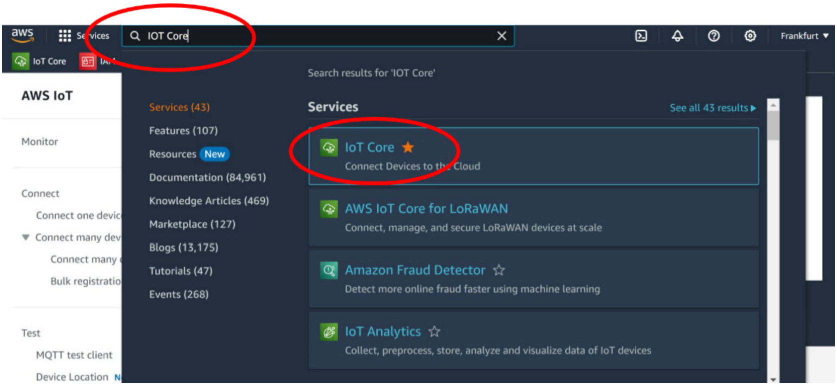

aws
Services
IOT Core
Frankfurt
IoT Core
AWS IoT
Monitor
Connect
Connect one devic
Connect many dev
Connect many
Bulk registratio
Test
MQTT test client
Device Location
Search results for 'IOT Core'
Services (43)
Features (107)
Resources New
Documentation (84,961)
Knowledge Articles (469)
Marketplace (127)
Blogs (13,175)
Tutorials (47)
Events (268)
Services
See all 43 results
IoT Core
Connect Devices to the Cloud
AWS IoT Core for LoRaWAN
Connect, manage, and secure LoRaWAN devices at scale
Amazon Fraud Detector
Detect more online fraud faster using machine learning
IoT Analytics
Collect, preprocess, store, analyze and visualize data of IoT devices

Figure 9.13 – IoT Core on the AWS console

AWS IoT uses *X.509 certificates* to enforce MQTT security. In the next sections, we are going to do the following:

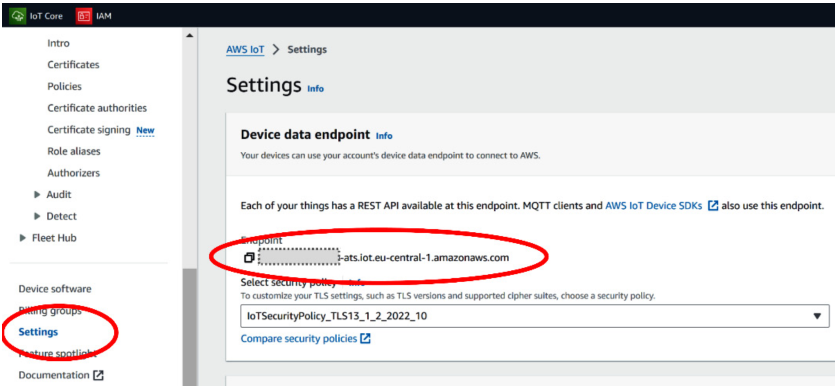
- Activate a policy to connect an external device through MQTT/MQTTS
- Create certificates
- Enable certificates and attach the previously created policy
- Use the certificates on the edge
- Connect the edge to the external data source
- Start sending data to IoT Core

Alternatively, we can achieve this in a single step: register a new device (which is called a thing by AWS) and connect it automatically.

After registration, we can implement our device on a local PC reusing our Node-RED implementation done in *Chapter 6* . In the final subsections, we will do the same with native AWS edge: Greengrass and SiteWise.

Identifying the AWS MQTT endpoint

Before starting our exercise, we need to know the AWS MQTT endpoint of our AWS instance. From Settings , locate the endpoint as depicted in the following figure and save it for the next step:


IoT Core
IAM
Intro
Certificates
Policies
Certificate authorities
Certificate signing New
Role aliases
Authorizers
Audit
Detect
Fleet Hub
Device software
Billing groups
Settings
Feature spotlight
Documentation
AWS IoT > Settings
Info
Info
Info
Compare security policies

Settings

Device data endpoint

Your devices can use your account's device data endpoint to connect to AWS.

Each of your things has a REST API available at this endpoint. MQTT clients and AWS IoT Device SDKs also use this endpoint.

Endpoint
[redacted]-ats.iot.eu-central-1.amazonaws.com

Select security policy

To customize your TLS settings, such as TLS versions and supported cipher suites, choose a security policy.

IoTSecurityPolicy_TLS13_1_2_2022_10

Figure 9.14 – Locating the AWS IoT endpoint

Having configured IoT Core, we will now connect our edges.

Connecting Node-RED

In *Chapter 6* , we learned about OPC UA. In this simple exercise, we want to connect our OPC UA server with the edge and send data to AWS IoT Core.

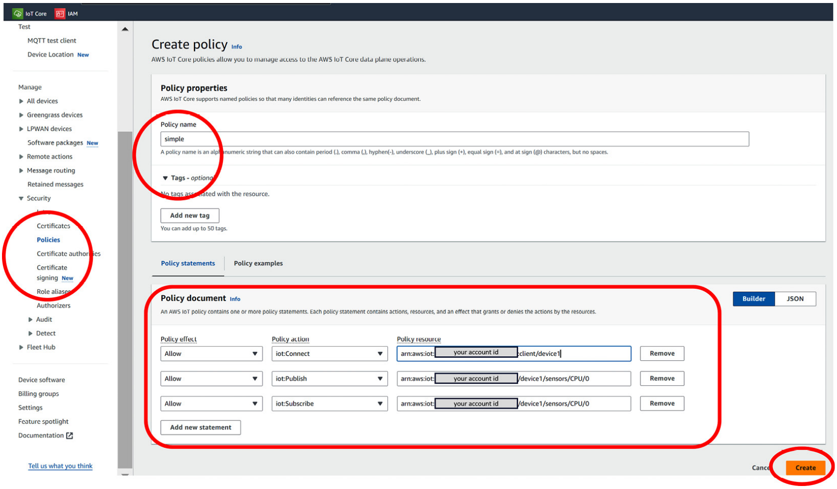
We are going first to reuse Node-RED and OPC UA-server used in the previous chapters and then we will replace this edge with a native AWS IoT edge client in the next section. The purpose of this exercise is to understand the flexibility and portability of AWS IoT. Indeed, it is not always possible to replace an edge fleet and we need to keep multi-version interoperability.

Setting the security policy for a Node-RED device

AWS IoT applies security to protect our system. To allow a device to send data, we need to attach the following statements to the device: `iot:Connect`, `iot:Subscribe`, `iot:Publish`, and `iot:Receive`.

We do this by creating a policy in AWS that includes these statements, as follows:

1. To create a policy, click on Security and then Policies , and register a new policy by clicking on the Create a policy button.
2. Then, provide the `iiot-book-policy-4-mqtt` name and add the `iot:Connect`, `iot:Subscribe`, and `iot:Publish` or `iot:*` statements (`iot:*` is strongly discouraged for security reasons). Allow the statements and click on the Create button to build the policy:


IoT Core
IAM
Test
MQTT test client
Device Location New
Manage
All devices
Greengrass devices
LPWAN devices
Software packages New
Remote actions
Message routing
Retained messages
Security
Certificates
Policies
Certificate authorities
Certificate signing New
Role aliases
Authorizers
Audit
Detect
Fleet Hub
Device software
Billing groups
Settings
Feature spotlight
Documentation
Tell us what you think
Info
Add new tag
Policy statements
Policy examples
Info
Builder
JSON
Remove
Add new statement
Cancel
Create

Create policy

AWS IoT Core policies allow you to manage access to the AWS IoT Core data plane operations.

Policy properties

AWS IoT Core supports named policies so that many identities can reference the same policy document.

Policy name: simple

A policy name is an alphanumeric string that can also contain period (.), comma (,), hyphen(-), underscore (_), plus sign (+), equal sign (=), and at sign (@) characters, but no spaces.

Tags - optional

No tags associated with the resource.

You can add up to 50 tags.

Policy document

An AWS IoT policy contains one or more policy statements. Each policy statement contains actions, resources, and an effect that grants or denies the actions by the resources.

Policy effect	Policy action	Policy resource
Allow	iot:Connect	arn:aws:iot: your account id :client/device1
Allow	iot:Publish	arn:aws:iot: your account id /device1/sensors/CPU/0
Allow	iot:Subscribe	arn:aws:iot: your account id /device1/sensors/CPU/0

Figure 9.15 – Creating a policy to enforce security

Here, the resource name is as follows:

```
arn:aws:iot:<your zone>:<your account
id>:/topic/device1/sensors/CPU/0
```

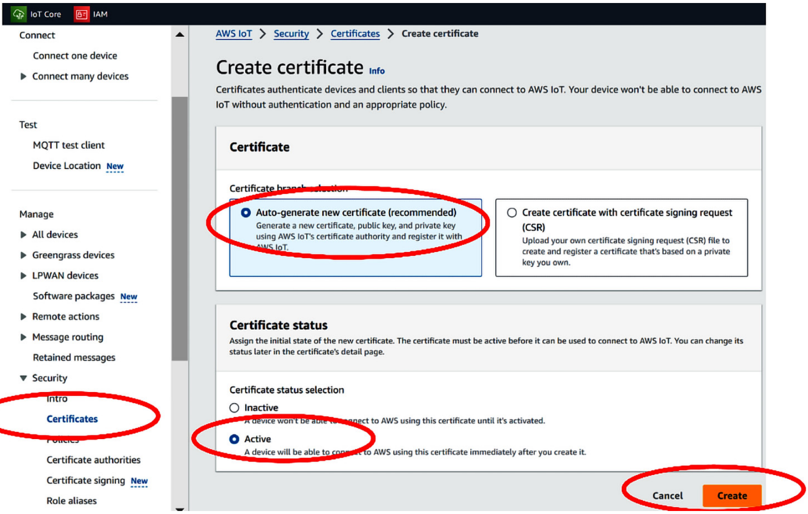
And for connect:

```
arn:aws:iot:<your zone>:<your account
id>:client/device1
```

Alternatively, we can use `*` to specify a simple policy. However, as we saw in *Chapter 6* , this is not recommended for security reasons:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish",
        "iot:Receive",
        "iot:PublishRetain"
      ],
      "Resource":
"arn:aws:<region>:<account>:topic/*"
    },
    {
      "Effect": "Allow",
      "Action": "iot:Subscribe",
      "Resource":
"arn:aws:<region>:<account>:topicfilter/
*"
    },
    {
      "Effect": "Allow",
      "Action": "iot:Connect",
      "Resource":
"arn:aws:iot:<region>:<account>:client/
*"
    }
  ]
}
```

1. Now, we need to create a certificate and activate and attach the previously created policy. From Security | Certificates , build an auto-generated certificate and activate it:


IoT Core
IAM
Connect
Connect one device
Connect many devices
Test
MQTT test client
Device Location New
Manage
All devices
Greengrass devices
LPWAN devices
Software packages New
Remote actions
Message routing
Retained messages
Security
Intro
Certificates
Policies
Certificate authorities
Certificate signing New
Role aliases
AWS IoT > Security > Certificates > Create certificate
Info
Cancel
Create

Create certificate

Certificates authenticate devices and clients so that they can connect to AWS IoT. Your device won't be able to connect to AWS IoT without authentication and an appropriate policy.

Certificate

Certificate branch selection

● Auto-generate new certificate (recommended)
Generate a new certificate, public key, and private key using AWS IoT's certificate authority and register it with AWS IoT.

○ Create certificate with certificate signing request (CSR)
Upload your own certificate signing request (CSR) file to create and register a certificate that's based on a private key you own.

Certificate status

Assign the initial state of the new certificate. The certificate must be active before it can be used to connect to AWS IoT. You can change its status later in the certificate's detail page.

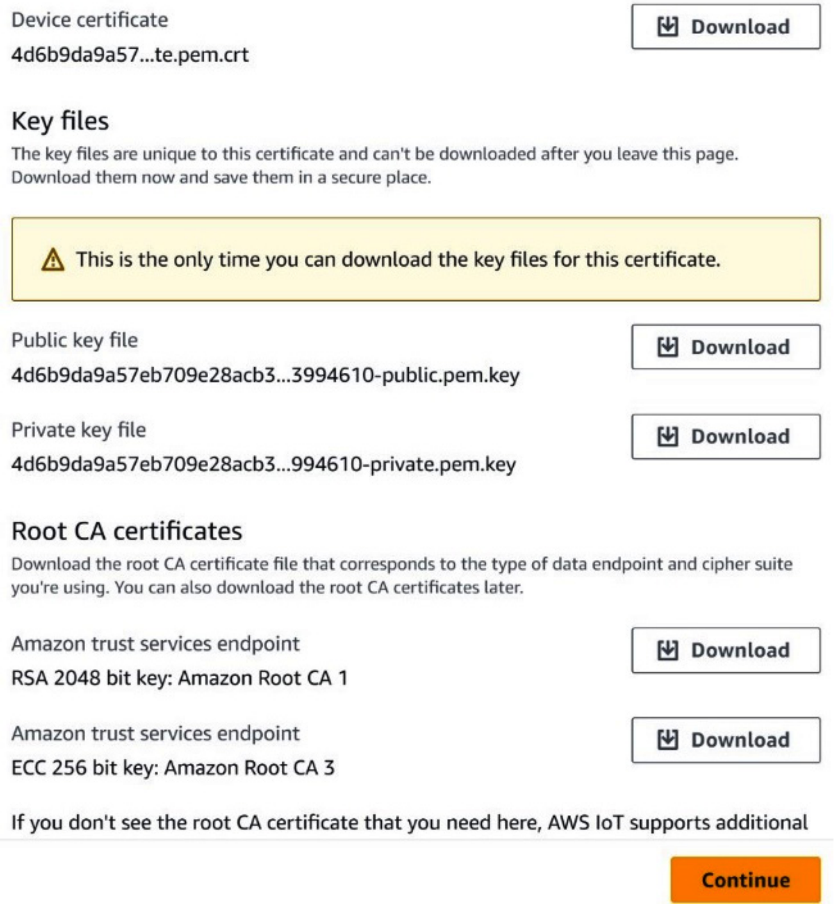
Certificate status selection

○ Inactive
A device won't be able to connect to AWS using this certificate until it's activated.

● Active
A device will be able to connect to AWS using this certificate immediately after you create it.

Figure 9.16 – Creating a certificate

After creation, download all certificates, including root CA certificates, as shown here:



Device certificate
4d6b9da9a57...te.pem.crt

Key files
The key files are unique to this certificate and can't be downloaded after you leave this page. Download them now and save them in a secure place.

⚠ This is the only time you can download the key files for this certificate.

Public key file
4d6b9da9a57eb709e28acb3...3994610-public.pem.key

Private key file
4d6b9da9a57eb709e28acb3...994610-private.pem.key

Root CA certificates
Download the root CA certificate file that corresponds to the type of data endpoint and cipher suite you're using. You can also download the root CA certificates later.

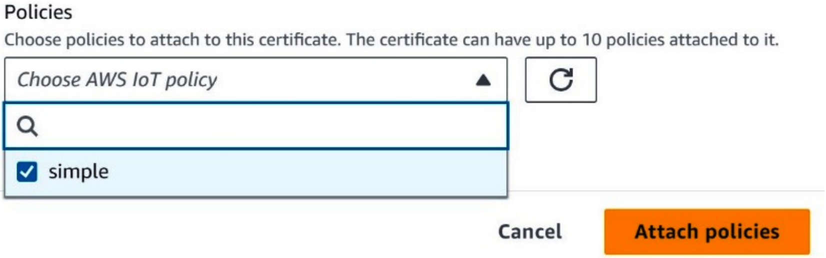
Amazon trust services endpoint
RSA 2048 bit key: Amazon Root CA 1

Amazon trust services endpoint
ECC 256 bit key: Amazon Root CA 3

If you don't see the root CA certificate that you need here, AWS IoT supports additional

Figure 9.17 – Downloading certificate

1. Finally, click on the certificate just created and attach the simple policy built during *step 1* :



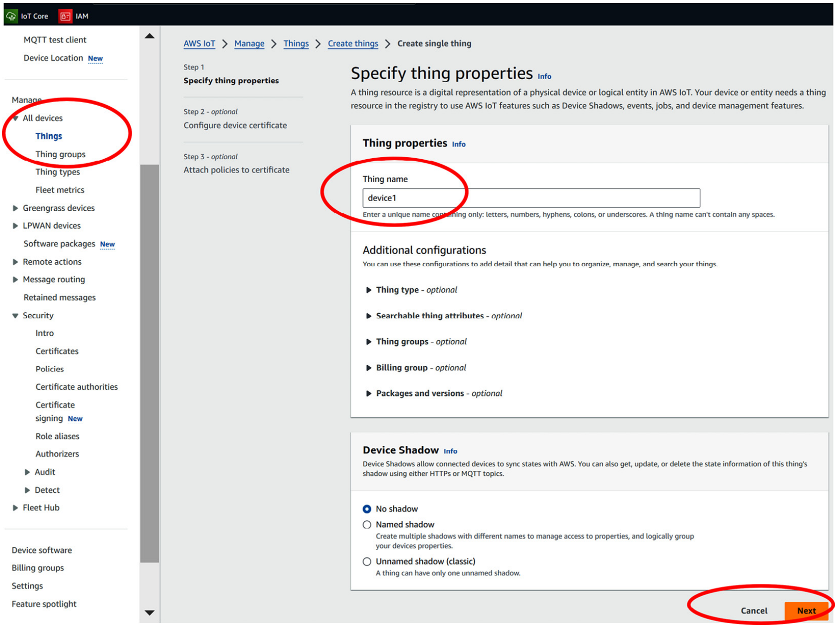
Policies
Choose policies to attach to this certificate. The certificate can have up to 10 policies attached to it.

Choose AWS IoT policy

☑ simple

Figure 9.18 – Attaching the policy to the certificate

1. Now, we need to create a thing. From Things, create a new thing with the name `device1` . Please note that the name must be the same as the resource name of the policy. Click Next :


IoT Core
IAM
MQTT test client
Device Location New
Manage
All devices
Things
Thing groups
Thing types
Fleet metrics
Greengrass devices
LPWAN devices
Software packages New
Remote actions
Message routing
Retained messages
Security
Intro
Certificates
Policies
Certificate authorities
Certificate signing New
Role aliases
Authorizers
Audit
Detect
Fleet Hub
Device software
Billing groups
Settings
Feature spotlight
AWS IoT > Manage > Things > Create things > Create single thing
Step 1
Specify thing properties
Step 2 - optional
Configure device certificate
Step 3 - optional
Attach policies to certificate

Specify thing properties Info

A thing resource is a digital representation of a physical device or logical entity in AWS IoT. Your device or entity needs a thing resource in the registry to use AWS IoT features such as Device Shadows, events, jobs, and device management features.

Thing properties Info

Thing name: device1

Enter a unique name containing only: letters, numbers, hyphens, colons, or underscores. A thing name can't contain any spaces.

Additional configurations

You can use these configurations to add detail that can help you to organize, manage, and search your things.

Thing type - optional
Searchable thing attributes - optional
Thing groups - optional
Billing group - optional
Packages and versions - optional

Device Shadow Info

Device Shadows allow connected devices to sync states with AWS. You can also get, update, or delete the state information of this thing's shadow using either HTTPs or MQTT topics.

● No shadow

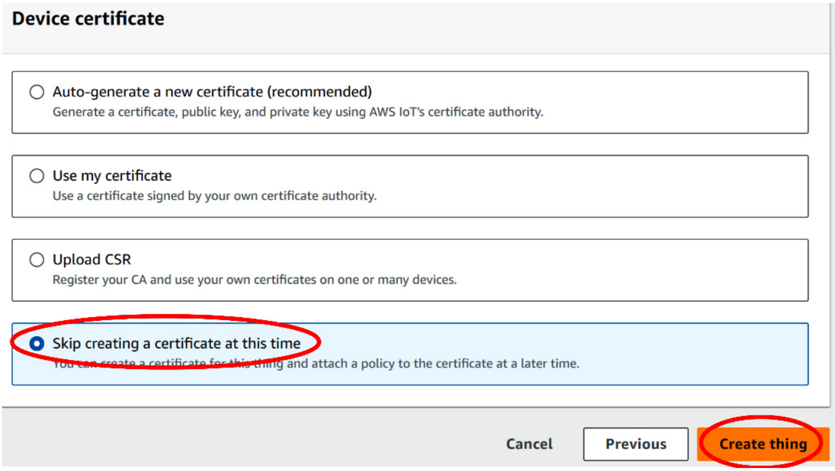
○ Named shadow
Create multiple shadows with different names to manage access to properties, and logically group your devices properties.

○ Unnamed shadow (classic)
A thing can have only one unnamed shadow.

Cancel
Next

Figure 9.19 – Creating a thing

We can skip the certificate creation because we have already done it:



Device certificate

○ Auto-generate a new certificate (recommended)
Generate a certificate, public key, and private key using AWS IoT's certificate authority.

○ Use my certificate
Use a certificate signed by your own certificate authority.

○ Upload CSR
Register your CA and use your own certificates on one or many devices.

● Skip creating a certificate at this time
You can create a certificate for this thing and attach a policy to the certificate at a later time.

Figure 9.20 – Skipping the certificate creation step

> ***Tip***

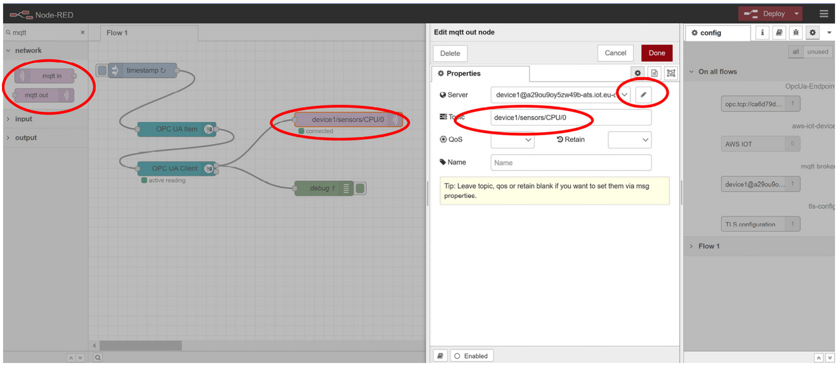
> *If you want to skip creating a policy manually, you can ask AWS to build the policy, build certificates, and activate them in a single step during the final step,* ***Create thing*** *, using the step recommended by AWS.*

Connecting Node-RED

We can reuse the Node-RED service implemented in the previous chapters. Start the Node-RED service by running the following command:

```
$ docker compose up
```

Now, we can connect to `http://localhost:1880/` and configure the OPC UA client and AWS MQTT client. Configure a new MQTT node as depicted in the following figure:



Edit mqtt out node

Server: device1@a29ou9oy5zw49b-ats.iot.eu-c

Topic: device1/sensors/CPU/0

Name: Name

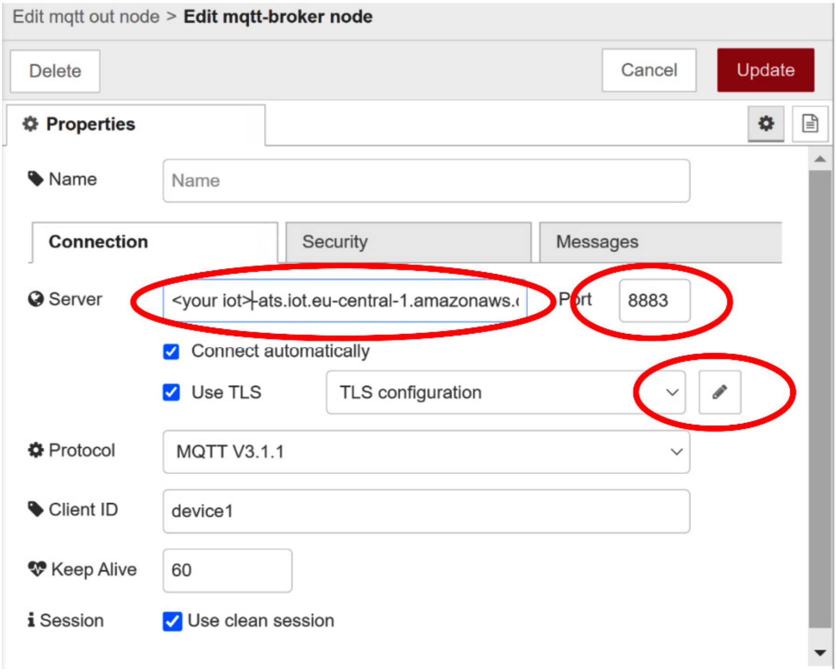
Tip: Leave topic, qos or retain blank if you want to set them via msg properties.

Node-RED
Deploy
mqtt
Flow 1
network
mqtt in
mqtt out
input
output
timestamp ↻
OPC UA Item
OPC UA Client
active reading
device1/sensors/CPU/0
connected
debug 1
Delete
Cancel
Done
Properties
QoS
Retain
Enabled
config
all
unused
On all flows
OpcUa-Endpoint
opc.tcp://ca6d79d...
1
aws-iot-device
AWS IOT
0
mqtt broker
device1@a29ou9o...
1
tls-config
TLS configuration
1
Flow 1

Figure 9.21 – Configuring a Node-RED MQTT node

For Topic , enter `device1/sensors/CPU/0` .

Then, click on Server and configure the AWS endpoint and the port to `8883` :


Edit mqtt out node > Edit mqtt-broker node
Delete
Cancel
Update
Properties
Connection
Security
Messages

Name: Name

Server: <your iot>-ats.iot.eu-central-1.amazonaws.

Port: 8883

☑ Connect automatically

☑ Use TLS: TLS configuration

Protocol: MQTT V3.1.1

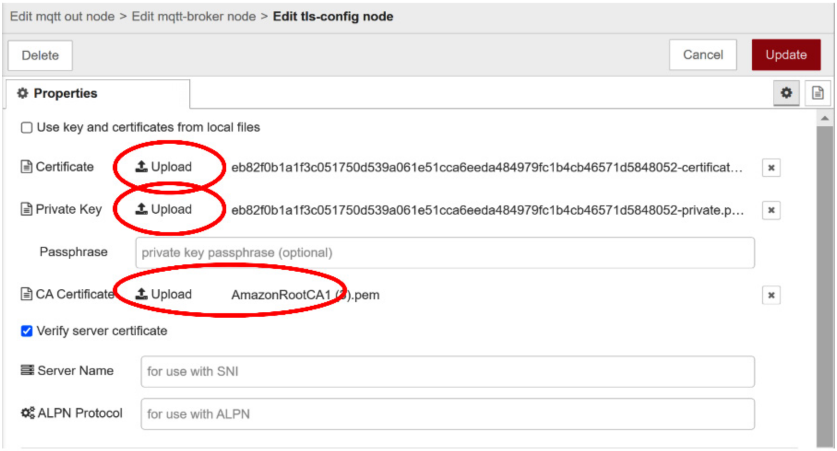
Client ID: device1

Keep Alive: 60

Session: ☑ Use clean session

Figure 9.22 – Configuring the AWS MQTT endpoint

Then, on TLS Configuration, upload the certificates downloaded before:


Edit mqtt out node > Edit mqtt-broker node > **Edit tls-config node**
Delete
Cancel
Update
Properties
Upload
Upload
Upload
private key passphrase (optional)
for use with SNI
for use with ALPN

☐ Use key and certificates from local files

Certificate: eb82f0b1a1f3c051750d539a061e51cca6eeda484979fc1b4cb46571d5848052-certificat...

Private Key: eb82f0b1a1f3c051750d539a061e51cca6eeda484979fc1b4cb46571d5848052-private.p...

Passphrase:

CA Certificate: AmazonRootCA1 (3).pem

☑ Verify server certificate

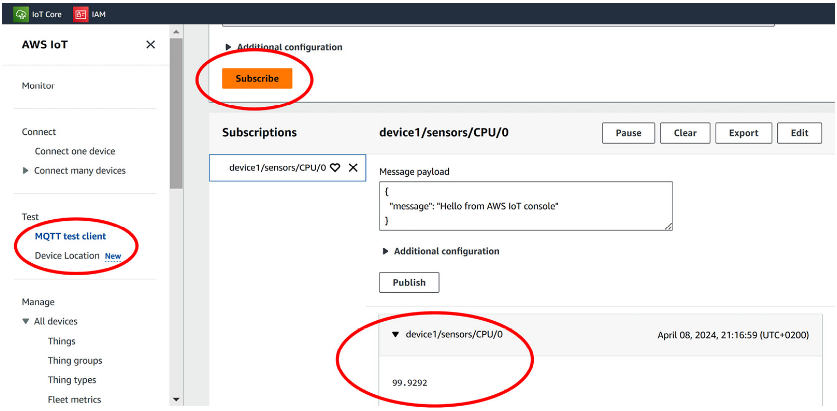
Server Name:

ALPN Protocol:

Figure 9.23 – Uploading certificates

Finally, deploy the flow. Node-RED will start sending the CPU information to AWS IoT Core; to verify it, from IoT Core, click MQTT test client and subscribe to `device1/sensors/CPU/0`.

If everything goes well, you will see the value of your CPU AWS IoT Core, as depicted in the following figure:


IoT Core
IAM
AWS IoT
Monitor
Connect
Connect one device
Connect many devices
Test
MQTT test client
Device Location New
Manage
All devices
Things
Thing groups
Thing types
Fleet metrics
Additional configuration
Subscribe
Subscriptions
Pause
Clear
Export
Edit
Message payload
Additional configuration
Publish

device1/sensors/CPU/0

```
{
  "message": "Hello from AWS IoT console"
}
```

device1/sensors/CPU/0 April 08, 2024, 21:16:59 (UTC+0200)

```
99.9292
```

Figure 9.24 – Subscribing to the MQTT topic

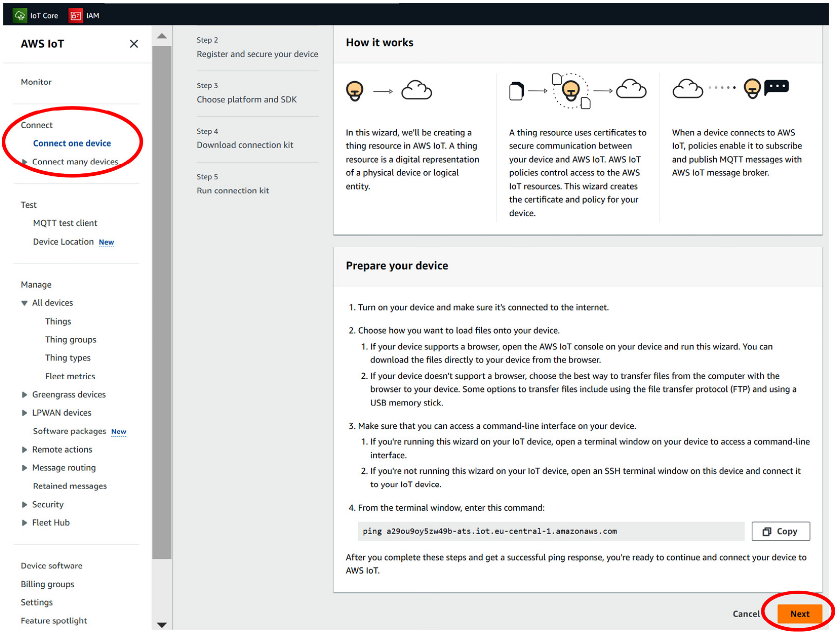
In this section, we configured our first edge device reusing our implementation of Node-RED. In the next section, we are going to implement our first edge device using the native AWS Edge SDK.

Using the AWS SDK

Node-RED is a client that is easy to configure, but in some cases, we need to integrate IoT Core on our edge system. AWS provides a useful SDK to work with Node.js, Python, Java, and C/C++. In this exercise, we will use the Python SDK to work with IoT Core. Follow these steps to configure the SDK:

1. First, we need to configure the thing. From the IoT console at https://

console.aws.amazon.com/iot/ , click on Connect one device and register a new device, as shown in the following screenshot:


IoT Core
IAM
AWS IoT
Monitor
Connect
Connect one device
Connect many devices
Test
MQTT test client
Device Location New
Manage
All devices
Things
Thing groups
Thing types
Fleet metrics
Greengrass devices
LPWAN devices
Software packages New
Remote actions
Message routing
Retained messages
Security
Fleet Hub
Device software
Billing groups
Settings
Feature spotlight
Step 2
Register and secure your device
Step 3
Choose platform and SDK
Step 4
Download connection kit
Step 5
Run connection kit
Copy
Cancel
Next

How it works

In this wizard, we'll be creating a thing resource in AWS IoT. A thing resource is a digital representation of a physical device or logical entity.

A thing resource uses certificates to secure communication between your device and AWS IoT. AWS IoT policies control access to the AWS IoT resources. This wizard creates the certificate and policy for your device.

When a device connects to AWS IoT, policies enable it to subscribe and publish MQTT messages with AWS IoT message broker.

Prepare your device

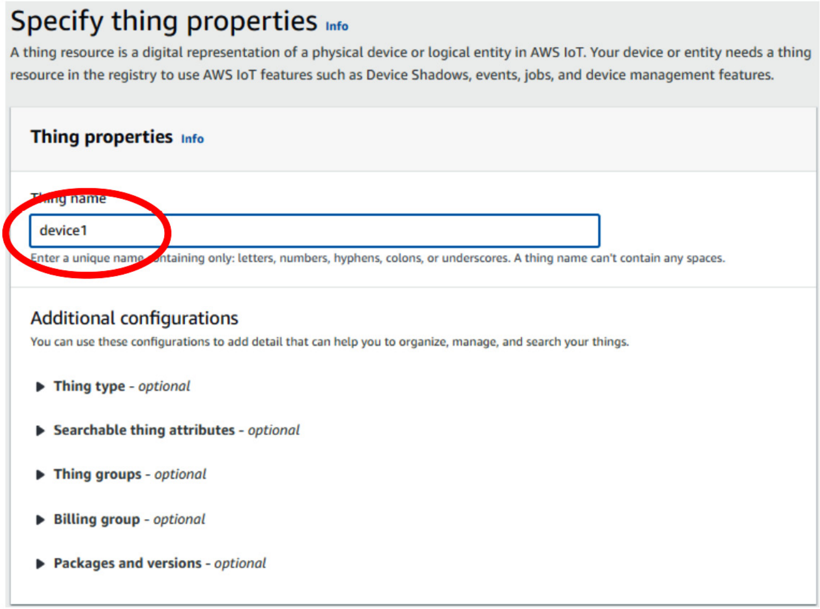
1. Turn on your device and make sure it's connected to the internet.
2. Choose how you want to load files onto your device.
   1. If your device supports a browser, open the AWS IoT console on your device and run this wizard. You can download the files directly to your device from the browser.
   2. If your device doesn't support a browser, choose the best way to transfer files from the computer with the browser to your device. Some options to transfer files include using the file transfer protocol (FTP) and using a USB memory stick.
3. Make sure that you can access a command-line interface on your device.
   1. If you're running this wizard on your IoT device, open a terminal window on your device to access a command-line interface.
   2. If you're not running this wizard on your IoT device, open an SSH terminal window on this device and connect it to your IoT device.
4. From the terminal window, enter this command:

```
ping a29ou9oy5zw49b-ats.iot.eu-central-1.amazonaws.com
```

After you complete these steps and get a successful ping response, you're ready to continue and connect your device to AWS IoT.

Figure 9.25 – Registering a new device

1. Click on Next . Then, enter your device name and click on the Next button:



Specify thing properties Info

A thing resource is a digital representation of a physical device or logical entity in AWS IoT. Your device or entity needs a thing resource in the registry to use AWS IoT features such as Device Shadows, events, jobs, and device management features.

Thing properties Info

Thing name

device1

Enter a unique name containing only: letters, numbers, hyphens, colons, or underscores. A thing name can't contain any spaces.

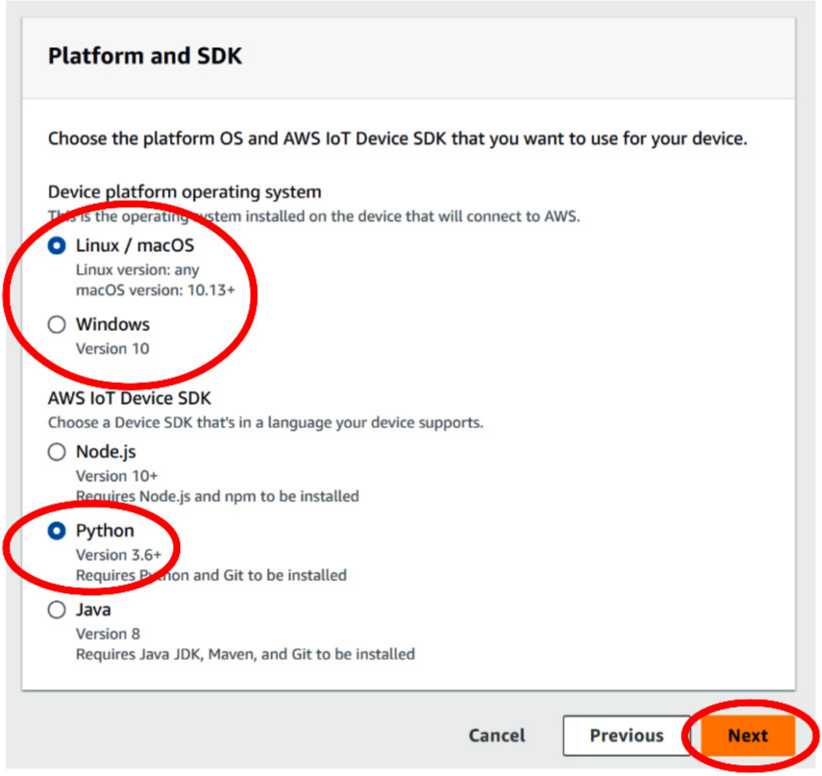
Additional configurations

You can use these configurations to add detail that can help you to organize, manage, and search your things.

▶ Thing type - optional
▶ Searchable thing attributes - optional
▶ Thing groups - optional
▶ Billing group - optional
▶ Packages and versions - optional

Figure 9.26 – Setting up our device in AWS

1. Now, select the SDK and platform and click Next :



Platform and SDK

Choose the platform OS and AWS IoT Device SDK that you want to use for your device.

Device platform operating system

This is the operating system installed on the device that will connect to AWS.

● Linux / macOS
Linux version: any
macOS version: 10.13+

○ Windows
Version 10

AWS IoT Device SDK

Choose a Device SDK that's in a language your device supports.

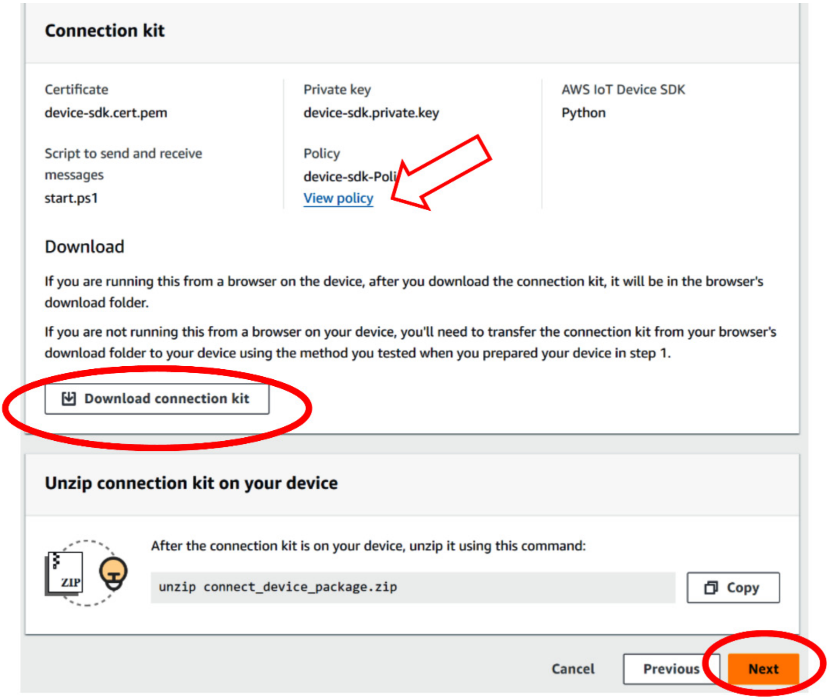
○ Node.js
Version 10+
Requires Node.js and npm to be installed

● Python
Version 3.6+
Requires Python and Git to be installed

○ Java
Version 8
Requires Java JDK, Maven, and Git to be installed

Figure 9.27 – SDK and platform

Now, AWS will create a policy, certificates, and script to download the SDK:



Connection kit

Certificate
device-sdk.cert.pem

Private key
device-sdk.private.key

AWS IoT Device SDK
Python

Script to send and receive messages
start.ps1

Policy
device-sdk-Poli
View policy

Download

If you are running this from a browser on the device, after you download the connection kit, it will be in the browser's download folder.

If you are not running this from a browser on your device, you'll need to transfer the connection kit from your browser's download folder to your device using the method you tested when you prepared your device in step 1.

Unzip connection kit on your device

After the connection kit is on your device, unzip it using this command:

```
unzip connect_device_package.zip
```

Figure 9.28 – Downloading the SDK

> ## *Policy*

> *As highlighted by the arrow in the previous figure, open the policy and check it: the device name **pubSub** is allowed to publish to the* `sdk/test/python` *topic. If you want to change the device name and topic, you have to reconfigure the policy.*

1. The final step is to subscribe to the IoT Core topic.

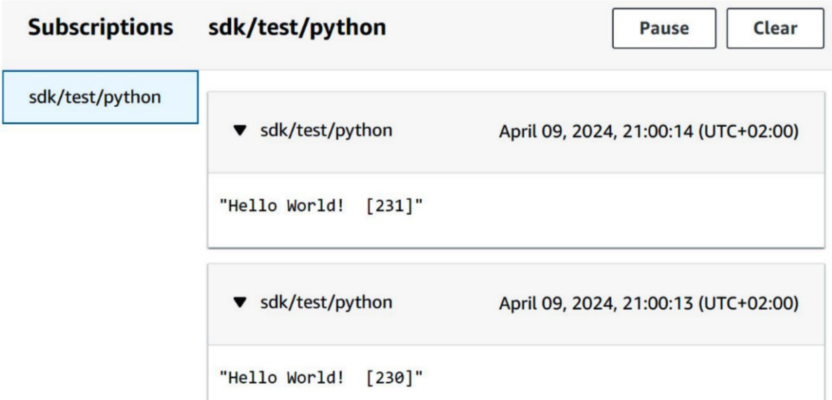
The device should be now ready and we can start it. Unzip the package just downloaded. From Windows PowerShell , on the downloaded folder, we can use the following code to start the edge:

```
$ Set-ExecutionPolicy -
ExecutionPolicy Bypass -Scope
Process
$ .\start.ps1
```

Alternatively, if your system is Linux-based, from the Linux Bash console, on the downloaded folder, execute the following script to start the edge:

```
$ .\start.sh
```

The AWS script will download the root CA certificate and SDK and, after a few seconds, will start publishing messages on AWS IoT Core. We can see data flowing into AWS IoT Core directly on the AWS IoT Core page:


Subscriptions
sdk/test/python
Pause
Clear
sdk/test/python
sdk/test/python
April 09, 2024, 21:00:14 (UTC+02:00)

"Hello World! [231]"

sdk/test/python
April 09, 2024, 21:00:13 (UTC+02:00)

"Hello World! [230]"

Figure 9.29 – AWS IoT Core messages

You can add more devices by clicking on the Create button, or you can manage the certificates from the Secure menu. Alternatively, you can define more groups and more devices from the Manage menu. Finally, you can also subscribe to or test the MQTT topic using the MQTT test client.

1. Now, you need to connect to the OPC UA server. First, install the `opcua` Python package. From the Windows or Linux CLI, run the following command:

   ```
   $ pip install opcua
   ```

   Now, you can connect to `opcua`. In the source code downloaded during step 3, open the `aws-iot-device-sdk-python-v2/samples/`

`pubsub.py` file.

2. At the top of the file, add the following lines to import the packages:

```
import sys, datetime
from opcua import Client
```

3. At line 74, add the following lines of code to connect the **`opcua`** server:

```
# connect opcua
client = Client("opc.tcp://
localhost:26543")
client.connect()
```

4. At line 134, configure the message to publish:

```
# Now getting a variable node using
its browse path
myvarC =
client.get_node("ns=1;s=CPU").get_value(
myvarT =
client.get_node("ns=1;s=Temperature").ge
message = {
    "CPU": myvarC,
    "Temperature": myvarT,
    "timestamp":
str(datetime.datetime.now(tz=datetime.ti
    "device" :
os.getenv('AWS_DEVICE_NAME',
'device1')
}
```

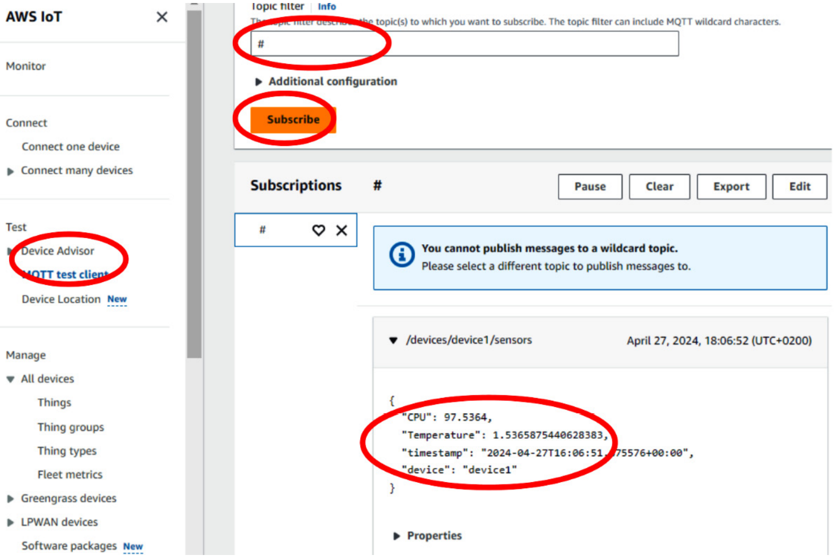
5. Restart the client.
6. From Windows PowerShell, in the downloaded folder, execute the following command to start the edge:

```
$ Set-ExecutionPolicy -
ExecutionPolicy Bypass -Scope
Process
$ .\start.ps1
```

7. From the Linux console, in the downloaded folder, execute the following script to start the edge:

```
$ .\start.sh
```

The new implemented client will send the local CPU percentage to IoT Core, as shown in the following figure:


AWS IoT
Monitor
Connect
Connect one device
Connect many devices
Test
Device Advisor
MQTT test client
Device Location New
Manage
All devices
Things
Thing groups
Thing types
Fleet metrics
Greengrass devices
LPWAN devices
Software packages New
Topic filter Info
#
Additional configuration
Subscribe
Subscriptions
#
Pause
Clear
Export
Edit
#
Properties

The topic filter describes the topic(s) to which you want to subscribe. The topic filter can include MQTT wildcard characters.

You cannot publish messages to a wildcard topic.
Please select a different topic to publish messages to.

/devices/device1/sensors April 27, 2024, 18:06:52 (UTC+0200)

```
{
  "CPU": 97.5364,
  "Temperature": 1.5365875440628383,
  "timestamp": "2024-04-27T16:06:51.[illegible]75576+00:00",
  "device": "device1"
}
```

Figure 9.30 – Subscribing to # to see the data sent to IoT Core

1. Clone the official book repository to get the final solution; from the CLI, run the following line:

```
$ git clone https://github.com/
PacktPublishing/Hands-On-
Industrial-Internet-of-Things-
Second-Edition
```

Greengrass V2

AWS Greengrass V2 extends AWS IoT cloud capabilities to local devices. Sometimes, it is not possible to process all the data on the cloud itself, so instead, we need to collect and analyze data closer to the source of information.

In other words, developers can use AWS Greengrass to run serverless code (AWS Lambda functions) in the cloud and conveniently deploy it to devices for the local execution of applications. AWS Greengrass also has great support for machine learning inference.

Greengrass is accessible from the menu on the left side of IoT Core under the Manage devices section.

AWS will deploy the package on the edge side, using the Greengrass runtime. The Greengrass runtime supports Java, Node.js, and Python. After deployment, Greengrass starts sending data to the cloud via the standard IoT Core endpoint. To configure Greengrass with OPC UA, we need to carry out the following steps:

1. Build the Greengrass v2 edge, the certificates, and the configuration file from the IoT Core interface.
2. Deploy and start Greengrass on the edge.

3. Build the OPC UA connector.

4. Deploy the OPC UA connector on the edge.

The following diagram shows the architecture of our exercise:

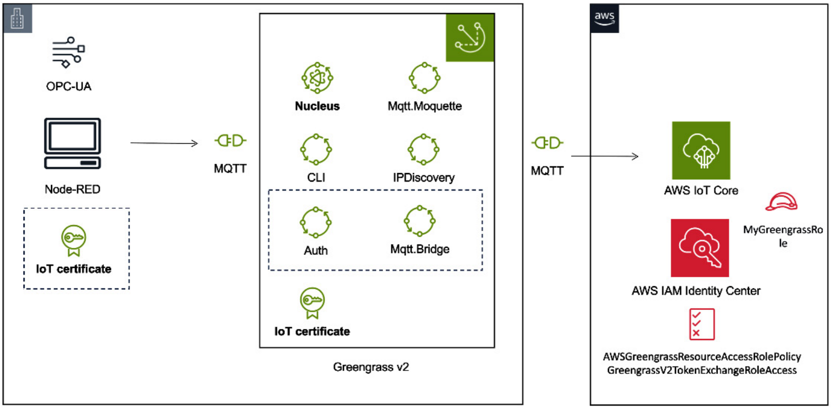
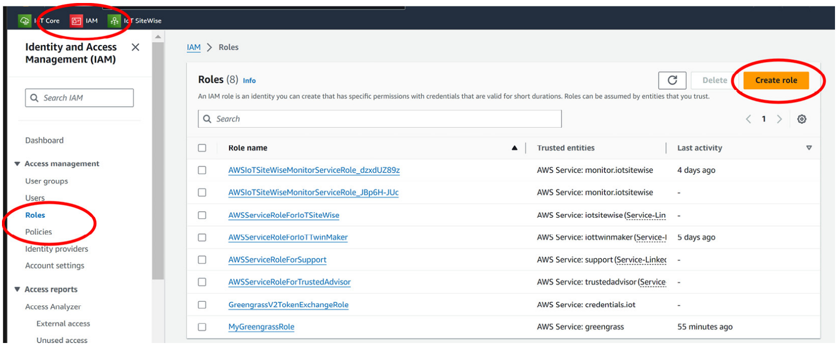

OPC-UA
Node-RED
IoT certificate
MQTT
Nucleus
Mqtt.Moquette
CLI
IPDiscovery
Auth
Mqtt.Bridge
IoT certificate
Greengrass v2
MQTT
aws
AWS IoT Core
MyGreengrassRole
AWS IAM Identity Center
AWSGreengrassResourceAccessRolePolicy
GreengrassV2TokenExchangeRoleAccess

Figure 9.31 – Architecture of the current exercise

Follow these steps:

1. Create the IAM role with the `AWSGreengrassResourceAccessRolePolicy` and `GreengrassV2TokenExchangeRoleAccess` policies assigned and attach them to IoT Core. During this first step, we need to attach the policies to AWS IoT:

    I. Connect to the IAM service and create a new role, as shown in the following figure:

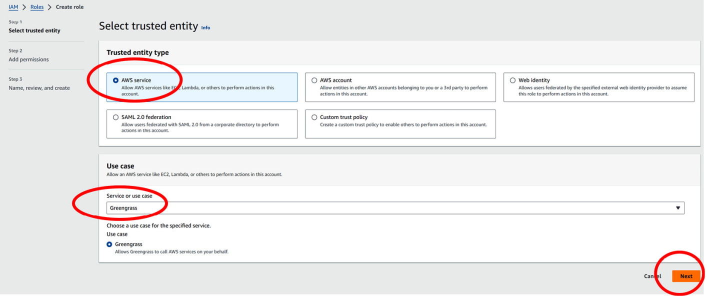

IoT Core
IAM
IoT SiteWise
Identity and Access Management (IAM)
Search IAM
Dashboard
Access management
User groups
Users
Roles
Policies
Identity providers
Account settings
Access reports
Access Analyzer
External access
Unused access
IAM > Roles
Roles (8) Info
Delete
Create role
Search
1

An IAM role is an identity you can create that has specific permissions with credentials that are valid for short durations. Roles can be assumed by entities that you trust.

☐	Role name	Trusted entities	Last activity
☐	AWSIoTSiteWiseMonitorServiceRole_dzxdUZ89z	AWS Service: monitor.iotsitewise	4 days ago
☐	AWSIoTSiteWiseMonitorServiceRole_JBp6H-JUc	AWS Service: monitor.iotsitewise	-
☐	AWSServiceRoleForIoTSiteWise	AWS Service: iotsitewise (Service-Lin	-
☐	AWSServiceRoleForIoTTwinMaker	AWS Service: iottwinmaker (Service-I	5 days ago
☐	AWSServiceRoleForSupport	AWS Service: support (Service-Linkec	-
☐	AWSServiceRoleForTrustedAdvisor	AWS Service: trustedadvisor (Service	-
☐	GreengrassV2TokenExchangeRole	AWS Service: credentials.iot	-
☐	MyGreengrassRole	AWS Service: greengrass	55 minutes ago

Figure 9.32 – Creating a new role for IoT Core

I. Provide a new name (e.g., `MyGreengrassRole`) and select the new trusted entity, as shown in the following figure:


IAM > Roles > Create role
Step 1
Select trusted entity
Step 2
Add permissions
Step 3
Name, review, and create
Select trusted entity Info
Trusted entity type

● AWS service
Allow AWS services like EC2, Lambda, or others to perform actions in this account.

○ AWS account
Allow entities in other AWS accounts belonging to you or a 3rd party to perform actions in this account.

○ Web identity
Allows users federated by the specified external web identity provider to assume this role to perform actions in this account.

○ SAML 2.0 federation
Allow users federated with SAML 2.0 from a corporate directory to perform actions in this account.

○ Custom trust policy
Create a custom trust policy to enable others to perform actions in this account.

Use case
Allow an AWS service like EC2, Lambda, or others to perform actions in this account.

Service or use case: Greengrass

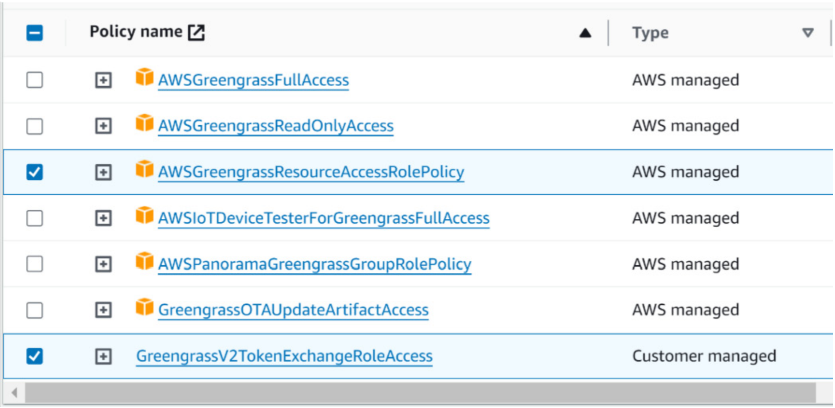
Choose a use case for the specified service.
Use case
● Greengrass
Allows Greengrass to call AWS services on your behalf.

Cancel
Next

Figure 9.33 – Configuring the entity

I. Then, attach the following policies:

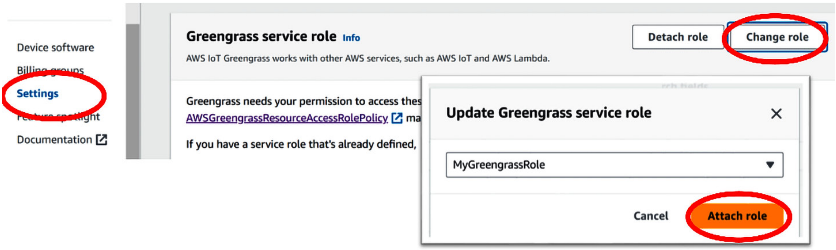
- AWSGreengrassResourceAccessRolePolicy
- GreengrassV2TokenExchangeRoleAccess



	Policy name	Type
☐	AWSGreengrassFullAccess	AWS managed
☐	AWSGreengrassReadOnlyAccess	AWS managed
☑	AWSGreengrassResourceAccessRolePolicy	AWS managed
☐	AWSIoTDeviceTesterForGreengrassFullAccess	AWS managed
☐	AWSPanoramaGreengrassGroupRolePolicy	AWS managed
☐	GreengrassOTAUpdateArtifactAccess	AWS managed
☑	GreengrassV2TokenExchangeRoleAccess	Customer managed

Figure 9.34 – Attaching new policies

I. Finally, in Settings in IoT Core, attach the just-created role:


Device software
Billing groups
Settings
Feature spotlight
Documentation
Info
Detach role
Change role
Cancel
Attach role

Greengrass service role

AWS IoT Greengrass works with other AWS services, such as AWS IoT and AWS Lambda.

Greengrass needs your permission to access thes
AWSGreengrassResourceAccessRolePolicy ma

If you have a service role that's already defined,

Update Greengrass service role

MyGreengrassRole

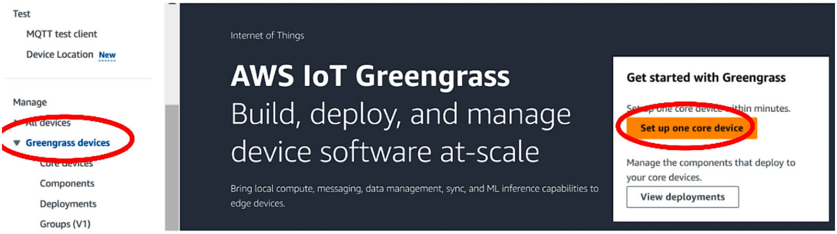
Figure 9.35 – Attaching the Greengrass role to IoT Core

This role allows AWS IoT Greengrass to verify client device identities. The role must specify AWS IoT Greengrass as a trusted entity and include the `AWSGreengrassResourceAccessRolePolicy` managed policy or custom policies that define equivalent permissions.

1. Now, we can move on to building the Greengrass edge:

    I. The first step to start with Greengrass is to configure the edge group on the IoT Core and download the Greengrass package, the certificates, and the configuration file.

    II. From the menu on the left, we need to click on Greengrass devices and follow the Get started with Greengrass procedure:


Test
MQTT test client
Device Location New
Manage
All devices
Greengrass devices
Core devices
Components
Deployments
Groups (V1)
Internet of Things
AWS IoT Greengrass
Get started with Greengrass
Set up one core device
View deployments

Build, deploy, and manage device software at-scale

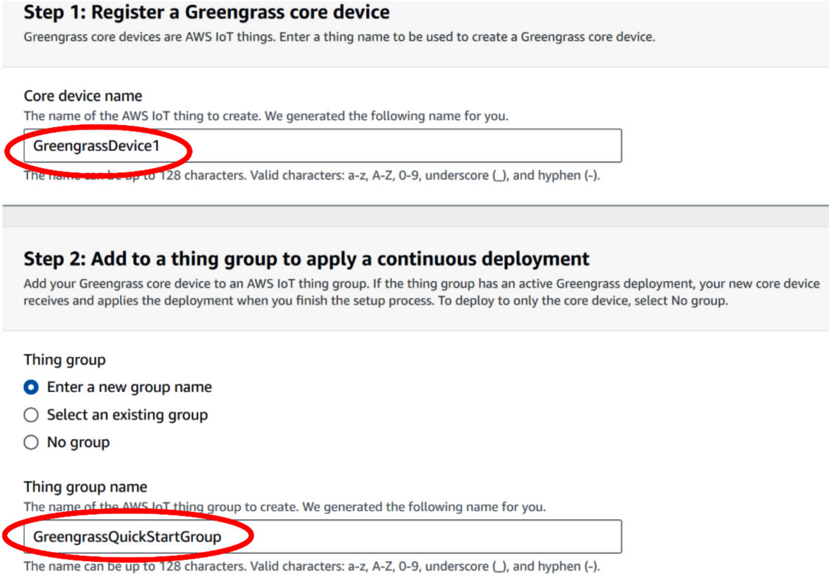
Bring local compute, messaging, data management, sync, and ML inference capabilities to edge devices.

Set up one core device within minutes.

Manage the components that deploy to your core devices.

Figure 9.36 – Building Greengrass v2 edge

I. The next step is to provide the name of the device, `GreenGrassDevice1`, and the name of the group, such as `GreengrassQuickStartGroup`, as shown in the following screenshot:



Step 1: Register a Greengrass core device

Greengrass core devices are AWS IoT things. Enter a thing name to be used to create a Greengrass core device.

Core device name

The name of the AWS IoT thing to create. We generated the following name for you.

GreengrassDevice1

The name can be up to 128 characters. Valid characters: a-z, A-Z, 0-9, underscore (_), and hyphen (-).

Step 2: Add to a thing group to apply a continuous deployment

Add your Greengrass core device to an AWS IoT thing group. If the thing group has an active Greengrass deployment, your new core device receives and applies the deployment when you finish the setup process. To deploy to only the core device, select No group.

Thing group

● Enter a new group name
○ Select an existing group
○ No group

Thing group name

The name of the AWS IoT thing group to create. We generated the following name for you.

GreengrassQuickStartGroup

The name can be up to 128 characters. Valid characters: a-z, A-Z, 0-9, underscore (_), and hyphen (-).

Figure 9.37 – Configuring Greengrass v2 edge

I. The installation suggests preparing the following script:

```
export
AWS_ACCESS_KEY_ID=<AWS_ACCESS_KEY_ID>
export
AWS_SECRET_ACCESS_KEY=<AWS_SECRET_ACCESS_KE
export
AWS_SESSION_TOKEN=<AWS_SESSION_TOKEN>
```

I. To get temporary credentials, run the following command:

```
$ aws sts get-session-token --
duration-seconds 1800
```

Tip

> *Alternatively, you can use CloudShell to get temporary credentials.*

> *Not recommended: you can use also the access key configured at the beginning of this chapter and leave* `AWS_SESSION_TOKEN` *blank.*

I. Then, download the Greengrass v2 distribution and the resources (certificates and configuration). Linux distribution is better supported but if you want to use the Windows version, you can follow the instructions in the official documentation: https://docs.aws.amazon.com/greengrass/v2/developerguide/getting-started-set-up-environment.html . We suggest using the Linux version and running it on Docker.

II. From the command line, run the following:

```
$ docker network create iiot-book-2-net-edge
$ docker run --rm -it --network iiot-book-2-net-edge amazonlinux:2023 /bin/bash
```

I. Then, install `unzip` and `curl` . From Docker Bash, run the following command to install `java` , `curl` , and `unzip` :

```
$ yum update && yum install -y unzip
```

```
wget sudo shadow-utils less systemd
systemd-sysv kernel-libbpf which
procps-ng java
```

You can download the installer using the `curl` command suggested by AWS and start Greengrass.

I. From the Bash console of Docker, follow these steps:

   i. Export the credentials.

   ii. Under Download the installer , choose Copy , and run the copied command on your device. This command downloads the latest version of the AWS IoT Greengrass Core software and unzips it on your device.

   iii. Under Run the installer , choose Copy , and run the copied command on your core device:

```
$ sudo –E java -Droot="/greengrass/v2"
-Dlog.store=FILE -jar ./
GreengrassInstaller/lib/Greengrass.jar
--aws-region eu-central-1 --thing-name
GreengrassDevice1 --thing-group-name
GreengrassQuickStartGroup --component-
default-user ggc_user:ggc_ group --
provision true --setup-system-service
true --deploy-dev-tools true
```

Please note that this command assumes `GreengrassDevice1` is deployed on `eu-`

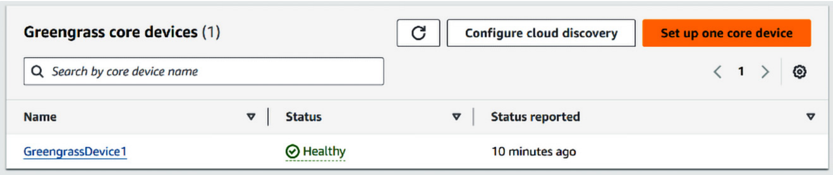
`central-1` so change it accordingly.

Workaround

If Greengrass is not installed as a service, you can set `--setup-system-service` *to* `false`*.*

These commands use the AWS IoT thing and thing group names that you specified earlier to run the AWS IoT Greengrass Core software installer and set up AWS resources for your core device.

Now, you should see GreenGrass communicating with IoT Core:


Configure cloud discovery
Set up one core device
Search by core device name
1

Greengrass core devices (1)

Name	Status	Status reported
GreengrassDevice1	Healthy	10 minutes ago

Figure 9.38 – Greengrass connected to IoT Core

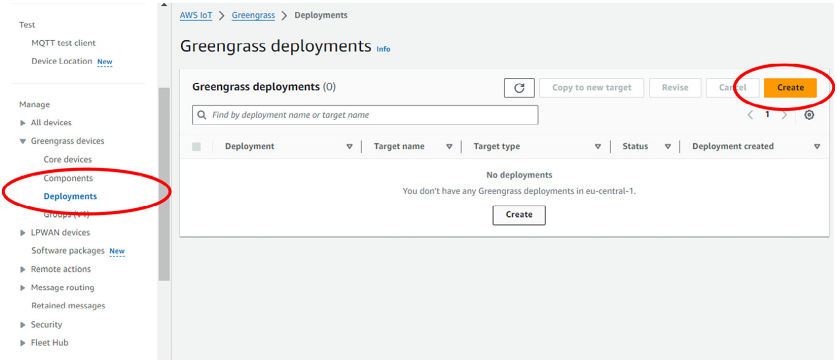
The code for this section is available in the official repository at https://github.com/PacktPublishing/Hands-On-Industrial-Internet-of-Things-Second-

Edition . You can use Dockerfile and `docker-compose` files. In the `docker-compose` file, change the placeholder with your access key.

1. Add additional components. The installation automatically deploys a basic component called **`Nucleus`** , which sends data to IoT Core.

   When the version of the Greengrass `nucleus` component changes, the AWS IoT Greengrass Core software—which includes the nucleus and all other components on your device—restarts to apply the changes.

2. From Deployments , create a new deployment:

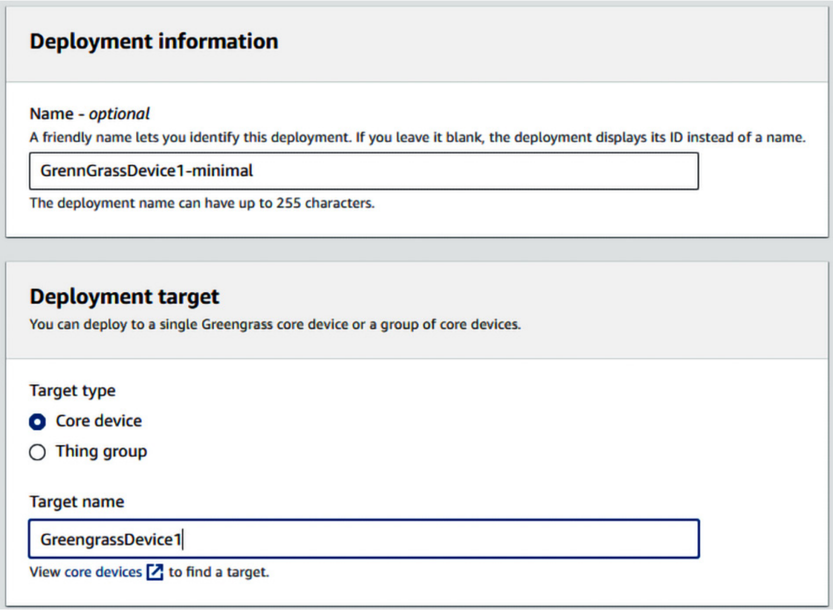

Test
MQTT test client
Device Location New
Manage
All devices
Greengrass devices
Core devices
Components
Deployments
Groups (V1)
LPWAN devices
Software packages New
Remote actions
Message routing
Retained messages
Security
Fleet Hub
AWS IoT > Greengrass > Deployments
Greengrass deployments Info
Greengrass deployments (0)
Copy to new target
Revise
Cancel
Create
Find by deployment name or target name
1
Deployment
Target name
Target type
Status
Deployment created
Create

No deployments

You don't have any Greengrass deployments in eu-central-1.

Figure 9.39 – Creating a new deployment

1. Then, provide a name and the target where you want to deploy the components; in our case, this is directly to the core device, as shown in the following figure:



Deployment information

Name - *optional*

A friendly name lets you identify this deployment. If you leave it blank, the deployment displays its ID instead of a name.

GrennGrassDevice1-minimal

The deployment name can have up to 255 characters.

Deployment target

You can deploy to a single Greengrass core device or a group of core devices.

Target type

● Core device

○ Thing group

Target name

GreengrassDevice1

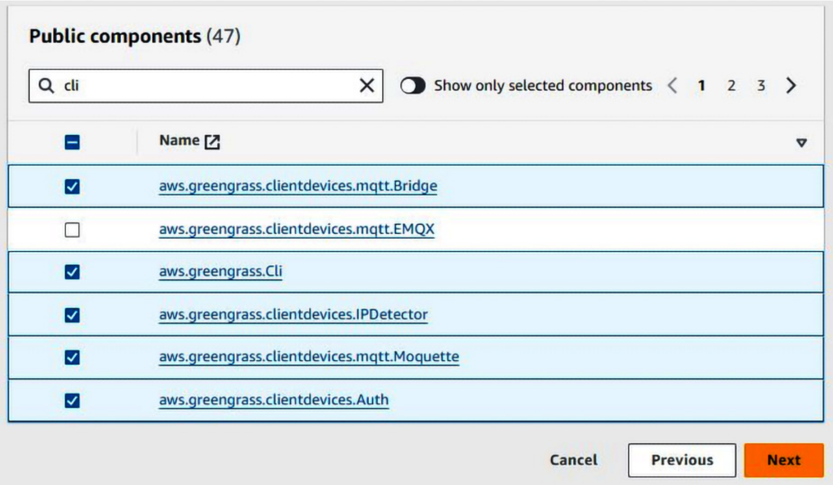
View core devices to find a target.

Figure 9.40 – Deploying basic components

1. Now, you can add the following components:

    - **`aws.greengrass.clientdevices.Auth`**
    - **`aws.greengrass.clientdevices.mqtt.M`**
    - **`aws.greengrass.clientdevices.mqtt.B`**
    - **`aws.greengrass.clientdevices.IPDete`**
    - **`aws.greengrass.Cli`**

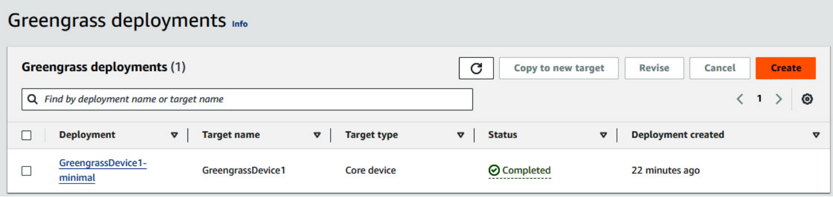
These are shown in the following figure:


Public components (47)
cli
Show only selected components
1 2 3
Cancel
Previous
Next

☐	Name
☑	aws.greengrass.clientdevices.mqtt.Bridge
☐	aws.greengrass.clientdevices.mqtt.EMQX
☑	aws.greengrass.Cli
☑	aws.greengrass.clientdevices.IPDetector
☑	aws.greengrass.clientdevices.mqtt.Moquette
☑	aws.greengrass.clientdevices.Auth

Figure 9.41 – Components to deploy on Greengrass

After the deployment (wait a few minutes) you should see the deployment correctly done, as shown in the following figure:


Greengrass deployments Info
Greengrass deployments (1)
Copy to new target
Revise
Cancel
Create
Find by deployment name or target name
1

☐	Deployment	Target name	Target type	Status	Deployment created
☐	GreengrassDevice1-minimal	GreengrassDevice1	Core device	✓ Completed	22 minutes ago

Figure 9.42 – Completed deployment

To verify the correctness of our procedure, connect to the Greengrass device and run the following command:

```
$ /greengrass/v2/bin/greengrass-cli
component list
```

You should see all the components installed.

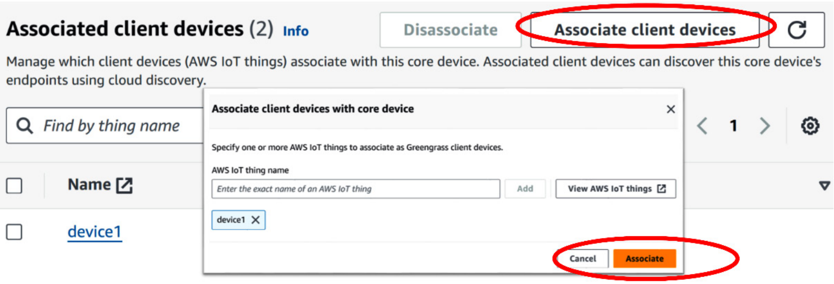
> ### *Tip on Docker*

> *If you are using Docker, you can open another console with* `docker exec -it mycontainer /bin/bash`*.*

In case of an error, check the logs:

```
$ /greengrass/v2/logs:
$ more /greengrass/v2/logs/*.log
```

1. Now, we can connect our external device through Greengrass to IOT Core. We should reuse the device created in the previous paragraph.
2. On `GreengrassDevice`, locate the Associated client devices tab and associate the client device, device1 , as depicted in the following figure:

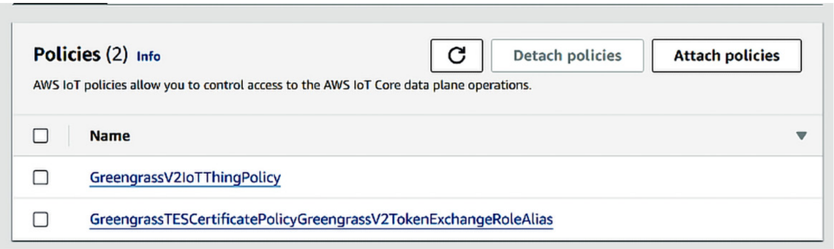

Associated client devices (2) Info
Disassociate
Associate client devices
Find by thing name
Name
device1
1
Associate client devices with core device
AWS IoT thing name
Enter the exact name of an AWS IoT thing
Add
View AWS IoT things
device1
Cancel
Associate

Manage which client devices (AWS IoT things) associate with this core device. Associated client devices can discover this core device's endpoints using cloud discovery.

Specify one or more AWS IoT things to associate as Greengrass client devices.

Figure 9.43 – Attaching the old device1 to GreengrassDevice

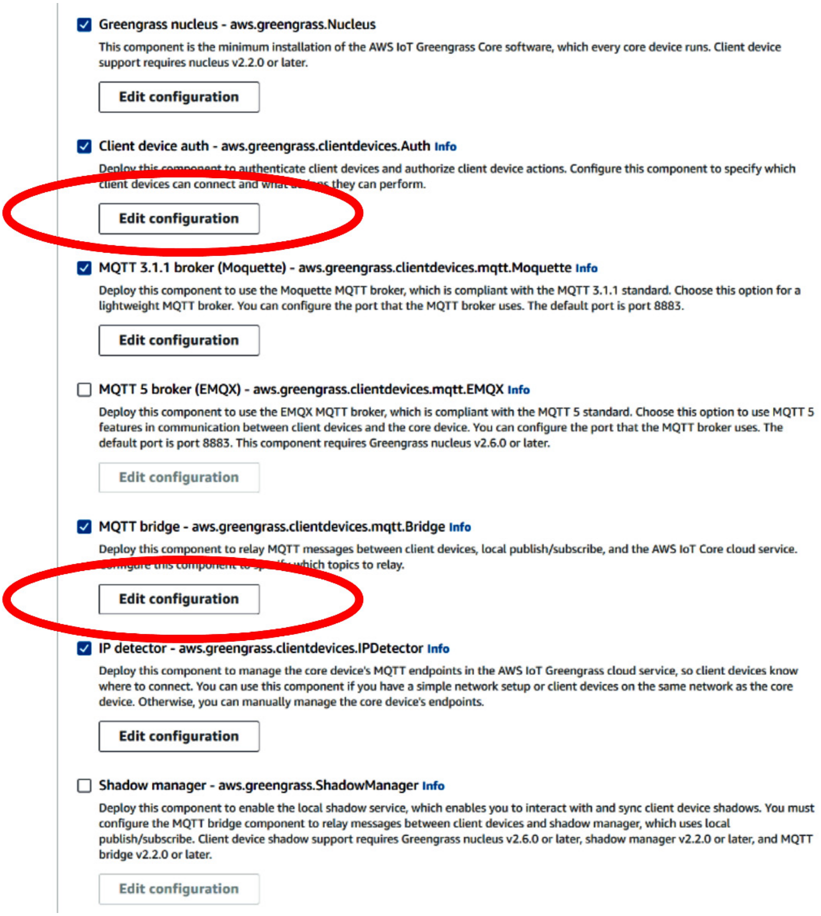
1. On the Certificates tab, ensure that the certificate of `device1` has been connected to the following policies:


Policies (2) Info
Detach policies
Attach policies
Name
GreengrassV2IoTThingPolicy
GreengrassTESCertificatePolicyGreengrassV2TokenExchangeRoleAlias

AWS IoT policies allow you to control access to the AWS IoT Core data plane operations.

Figure 9.44 – Attaching policies to the certificate

1. Click on Configure core device discovery . On the configuration component page, select the Auth component and click on Edit configuration :



☑ Greengrass nucleus - aws.greengrass.Nucleus

This component is the minimum installation of the AWS IoT Greengrass Core software, which every core device runs. Client device support requires nucleus v2.2.0 or later.

☑ Client device auth - aws.greengrass.clientdevices.Auth Info

Deploy this component to authenticate client devices and authorize client device actions. Configure this component to specify which client devices can connect and what actions they can perform.

☑ MQTT 3.1.1 broker (Moquette) - aws.greengrass.clientdevices.mqtt.Moquette Info

Deploy this component to use the Moquette MQTT broker, which is compliant with the MQTT 3.1.1 standard. Choose this option for a lightweight MQTT broker. You can configure the port that the MQTT broker uses. The default port is port 8883.

☐ MQTT 5 broker (EMQX) - aws.greengrass.clientdevices.mqtt.EMQX Info

Deploy this component to use the EMQX MQTT broker, which is compliant with the MQTT 5 standard. Choose this option to use MQTT 5 features in communication between client devices and the core device. You can configure the port that the MQTT broker uses. The default port is port 8883. This component requires Greengrass nucleus v2.6.0 or later.

☑ MQTT bridge - aws.greengrass.clientdevices.mqtt.Bridge Info

Deploy this component to relay MQTT messages between client devices, local publish/subscribe, and the AWS IoT Core cloud service. Configure this component to specify which topics to relay.

☑ IP detector - aws.greengrass.clientdevices.IPDetector Info

Deploy this component to manage the core device's MQTT endpoints in the AWS IoT Greengrass cloud service, so client devices know where to connect. You can use this component if you have a simple network setup or client devices on the same network as the core device. Otherwise, you can manually manage the core device's endpoints.

☐ Shadow manager - aws.greengrass.ShadowManager Info

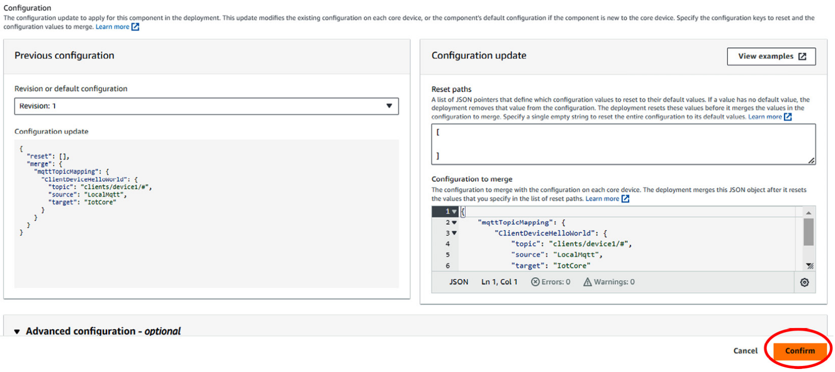
Deploy this component to enable the local shadow service, which enables you to interact with and sync client device shadows. You must configure the MQTT bridge component to relay messages between client devices and shadow manager, which uses local publish/subscribe. Client device shadow support requires Greengrass nucleus v2.6.0 or later, shadow manager v2.2.0 or later, and MQTT bridge v2.2.0 or later.

Figure 9.45 – Configuring components

1. On merge configuration, add the following JSON:

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName:
*",
        "policyName":
"MyPermissivePolicy"
      }
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
          "statementDescription":
"Allow client devices to perform all
actions.",
          "operations": [
            "*"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
}
```

See the following figure:



Configuration

The configuration update to apply for this component in the deployment. This update modifies the existing configuration on each core device, or the component's default configuration if the component is new to the core device. Specify the configuration keys to reset and the configuration values to merge. Learn more

Previous configuration

Revision or default configuration: Revision: 1

Configuration update

```
{
  "reset": [],
  "merge": {
    "mqttTopicMapping": {
      "ClientDeviceHelloWorld": {
        "topic": "clients/device1/#",
        "source": "LocalMqtt",
        "target": "IotCore"
      }
    }
  }
}
```

Configuration update

Reset paths

A list of JSON pointers that define which configuration values to reset to their default values. If a value has no default value, the deployment removes that value from the configuration. The deployment resets these values before it merges the values in the configuration to merge. Specify a single empty string to reset the entire configuration to its default values. Learn more

```
[

]
```

Configuration to merge

The configuration to merge with the configuration on each core device. The deployment merges this JSON object after it resets the values that you specify in the list of reset paths. Learn more

```
{
    "mqttTopicMapping": {
        "ClientDeviceHelloWorld": {
            "topic": "clients/device1/#",
            "source": "LocalMqtt",
            "target": "IotCore"
```

JSON Ln 1, Col 1 Errors: 0 Warnings: 0

Figure 9.46 – Clicking Confirm after adding the code

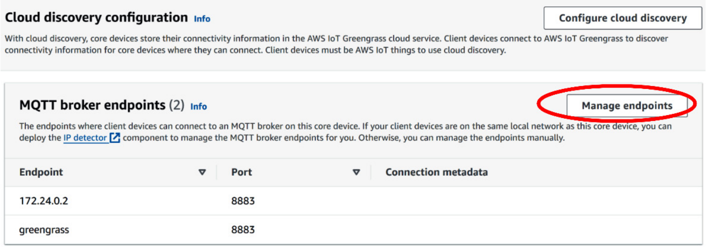
Repeat the same process for the MQTT `Bridge` component:

```
{
  "mqttTopicMapping": {
    "ClientDevice: {
      "topic": "clients/device1/#",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

These configurations allow every client to connect to Greengrass to redirect the data to IoT Core.

On the MQTT broker endpoint, add the following:

- IP address of the Greengrass server (auto-discovered by the IPDecetro component)
- Docker name of the container



Cloud discovery configuration Info

With cloud discovery, core devices store their connectivity information in the AWS IoT Greengrass cloud service. Client devices connect to AWS IoT Greengrass to discover connectivity information for core devices where they can connect. Client devices must be AWS IoT things to use cloud discovery.

MQTT broker endpoints (2) Info

The endpoints where client devices can connect to an MQTT broker on this core device. If your client devices are on the same local network as this core device, you can deploy the IP detector component to manage the MQTT broker endpoints for you. Otherwise, you can manage the endpoints manually.

Endpoint	Port	Connection metadata
172.24.0.2	8883	
greengrass	8883	

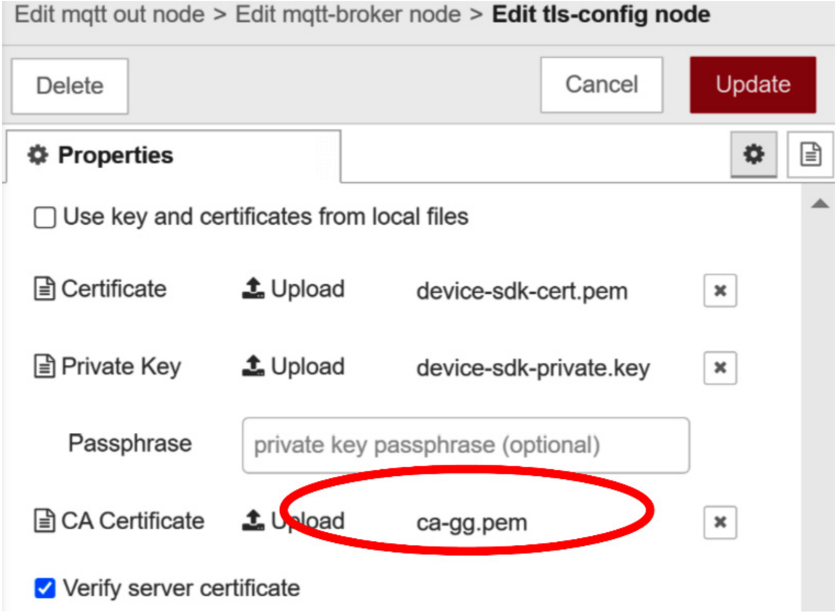
Figure 9.47 – MQTT endpoint

1. Revise the deployment and click on Deploy .

Now, we need a small trick; we need the CA of our Greengrass device. From the Bash console where you have installed Greengrass, extract the CA with the following command:

```
$ more /greengrass/v2/work/aws.greengrass.clientdevices.Auth/ca.pem
```

2. Copy and paste the CA certificate on a PEM file (e.g., `Ca-gg.pem`) and load it on the MQTT node of Node-RED:


Edit mqtt out node > Edit mqtt-broker node > Edit tls-config node
Delete
Cancel
Update
Properties

☐ Use key and certificates from local files

Certificate Upload device-sdk-cert.pem

Private Key Upload device-sdk-private.key

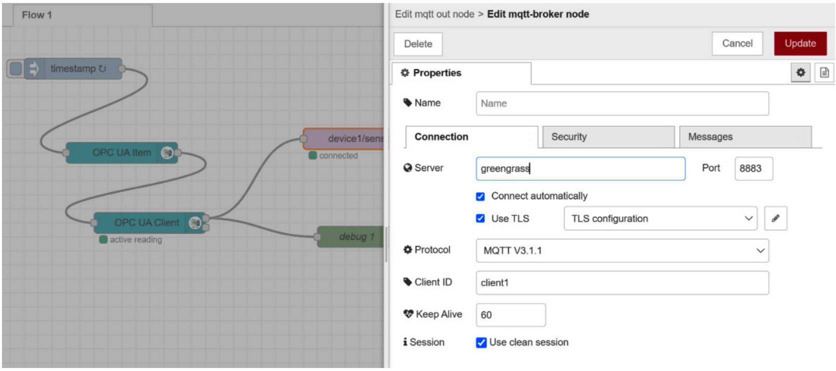
Passphrase private key passphrase (optional)

CA Certificate Upload ca-gg.pem

☑ Verify server certificate

Figure 9.48 – MQTT node TLS – updating the CA certificate

1. Finally, review the `node-red` client installed in the previous paragraph and connect it through the Greengrass device. Change the name of the server from IoT Core to Greengrass, as shown in the following figure:


Flow 1
timestamp
OPC UA Item
OPC UA Client
active reading
device1/sens
connected
debug 1
Edit mqtt out node > Edit mqtt-broker node
Delete
Cancel
Update
Properties
Connection
Security
Messages

Name: Name

Server: greengrass Port: 8883

☑ Connect automatically

☑ Use TLS: TLS configuration

Protocol: MQTT V3.1.1

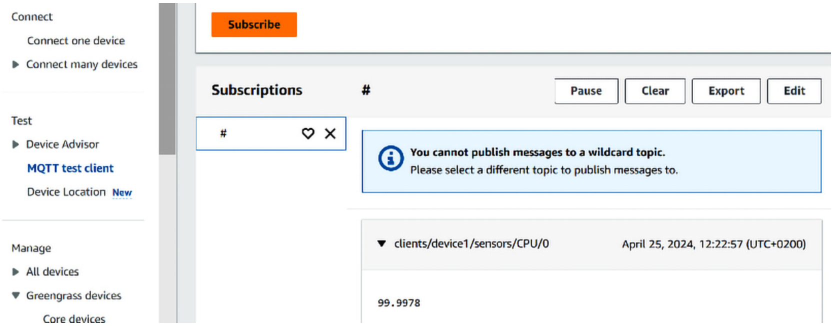
Client ID: client1

Keep Alive: 60

Session: ☑ Use clean session

Figure 9.49 – Redirecting Node-RED to the local Greengrass data broker

1. Now, from MQTT test client , subscribe to # . You can see the data sent to the cloud, as follows:


Connect
Connect one device
Connect many devices
Test
Device Advisor
MQTT test client
Device Location New
Manage
All devices
Greengrass devices
Core devices
Subscribe
Subscriptions
#
Pause
Clear
Export
Edit
#

You cannot publish messages to a wildcard topic.
Please select a different topic to publish messages to.

```
clients/device1/sensors/CPU/0    April 25, 2024, 12:22:57 (UTC+0200)

99.9978
```

Figure 9.50 – Data is sent to the cloud through Greengrass

You can also use the `python-sdk` client developed in the previous section.

About Greengrass

Why use Greengrass and not directly connect the device to IoT Core? It is common, in industrial devices, to place a data broker between the peripheral devices and the IoT platform capable of saving data locally in the event of disconnection (store and forward), deploying analytics on the edge, and enforcing cybersecurity (see *Chapter 5*).

Other interesting features of IIoT are the following:

- **`aws.greengrass.DiskSpooler`** supports saving data on local storage to send to IoT Core
- Device shadows : Shadows can make a device's state available to apps and other services, whether the device is connected to AWS IoT or not
- Lambda runtimes : Lambda allows to execute action on top of Greengrass

SiteWise

SiteWise is a new component for working with OPC UA and is specifically built for IIoT. In fact, OPC-UA is the standard protocol for industrial equipment and AWS has decided to develop a specific component for industrial purposes. We could say that SiteWise is the answer for the IoT industry. In this exercise, we will develop SiteWise on top of Greengrass.

To access SiteWise, search for it on the Services bar, as shown in the following figure:

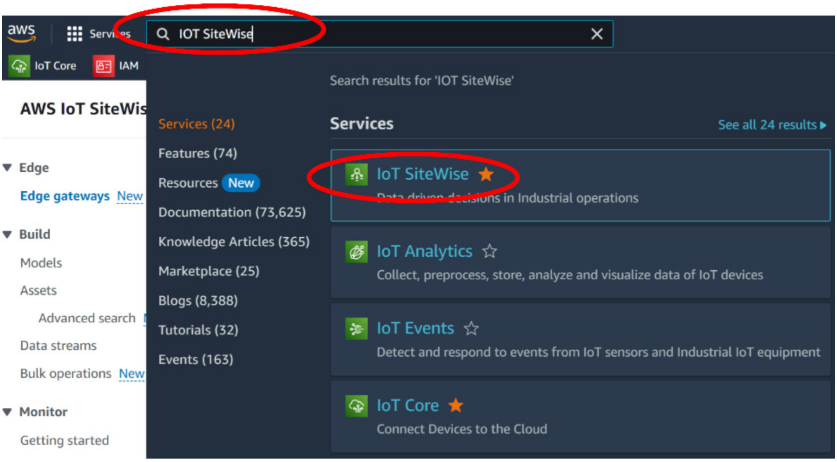
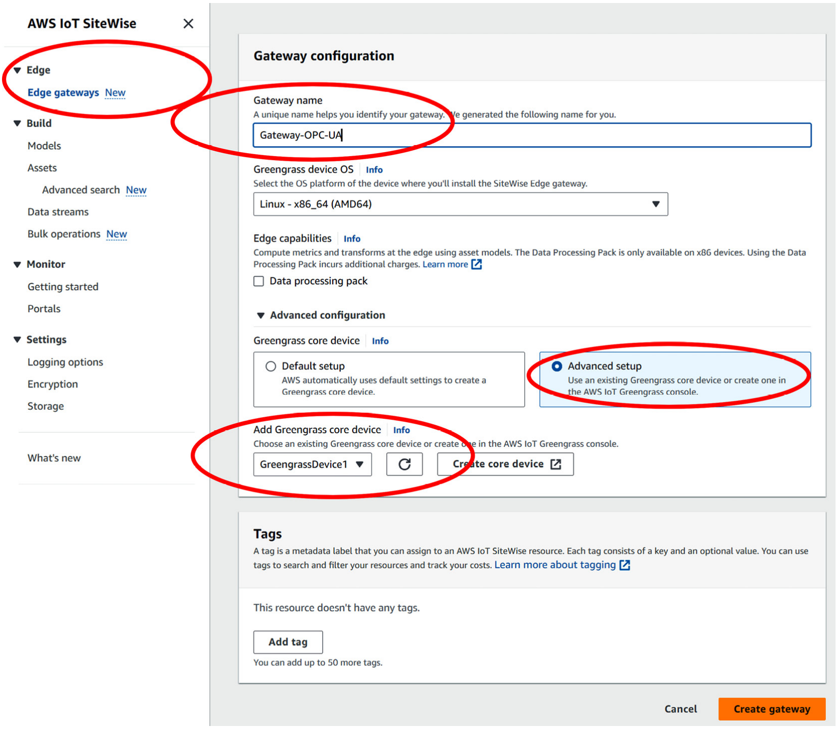

aws
Services
IOT SiteWise
IoT Core
IAM
AWS IoT SiteWis
Edge
Edge gateways New
Build
Models
Assets
Advanced search
Data streams
Bulk operations New
Monitor
Getting started
Search results for 'IOT SiteWise'
Services (24)
Features (74)
Resources New
Documentation (73,625)
Knowledge Articles (365)
Marketplace (25)
Blogs (8,388)
Tutorials (32)
Events (163)
Services
See all 24 results
IoT SiteWise
Data driven decisions in Industrial operations
IoT Analytics
Collect, preprocess, store, analyze and visualize data of IoT devices
IoT Events
Detect and respond to events from IoT sensors and Industrial IoT equipment
IoT Core
Connect Devices to the Cloud

Figure 9.51 – Locating SiteWise

Now, we are going to deploy the component to connect the Greengrass device directly to our `opcua` server. Indeed, SiteWise is a special component (`aws.iot.SiteWiseEdgePublisher`) of Greengrass. Follow these steps:

1. Build the component : From the left menu, create a new device and provide a name for it (e.g., `Gateway-OPC-UA`), choose Advanced setup , and deploy on the Greengrass device we created before:


AWS IoT SiteWise
Edge
Edge gateways New
Build
Models
Assets
Advanced search New
Data streams
Bulk operations New
Monitor
Getting started
Portals
Settings
Logging options
Encryption
Storage
What's new

Gateway configuration

Gateway name

A unique name helps you identify your gateway. We generated the following name for you.

Gateway-OPC-UA

Greengrass device OS Info

Select the OS platform of the device where you'll install the SiteWise Edge gateway.

Linux - x86_64 (AMD64)

Edge capabilities Info

Compute metrics and transforms at the edge using asset models. The Data Processing Pack is only available on x86 devices. Using the Data Processing Pack incurs additional charges. Learn more

☐ Data processing pack

Advanced configuration

Greengrass core device Info

○ Default setup
AWS automatically uses default settings to create a Greengrass core device.

● Advanced setup
Use an existing Greengrass core device or create one in the AWS IoT Greengrass console.

Add Greengrass core device Info

Choose an existing Greengrass core device or create one in the AWS IoT Greengrass console.

GreengrassDevice1

Tags

A tag is a metadata label that you can assign to an AWS IoT SiteWise resource. Each tag consists of a key and an optional value. You can use tags to search and filter your resources and track your costs. Learn more about tagging

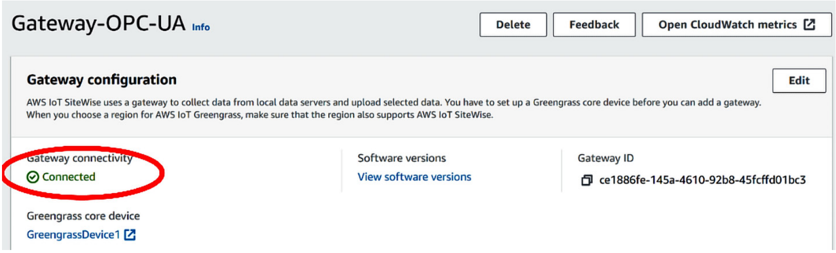
This resource doesn't have any tags.

You can add up to 50 more tags.

Create core device
Add tag
Cancel
Create gateway

Figure 9.52 – Building a SiteWise gateway

Wait a few minutes and check the status of the deployment:


Gateway-OPC-UA Info
Delete
Feedback
Open CloudWatch metrics
Edit

Gateway configuration

AWS IoT SiteWise uses a gateway to collect data from local data servers and upload selected data. You have to set up a Greengrass core device before you can add a gateway. When you choose a region for AWS IoT Greengrass, make sure that the region also supports AWS IoT SiteWise.

Gateway connectivity: Connected

Software versions: View software versions

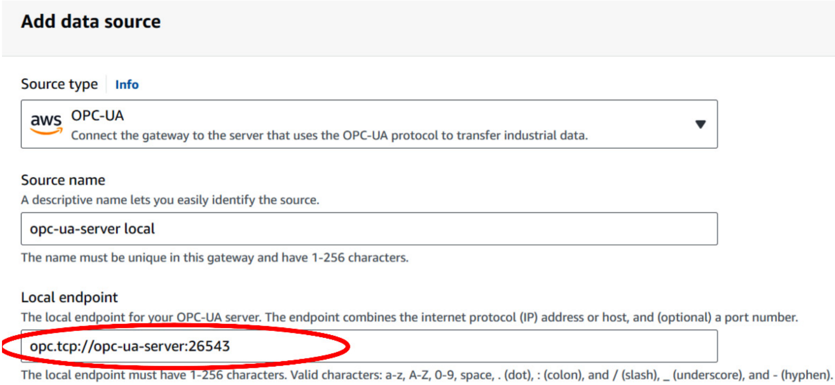
Gateway ID: ce1886fe-145a-4610-92b8-45fcffd01bc3

Greengrass core device: GreengrassDevice1

Figure 9.53 – Gateway connected

1. Connect to the OPC-UA server : Now, we can reuse our simulated OPC-UA server developed in the previous chapters. Click on the link

of the gateway just created and locate the data source section, then create a new data source. Provide a name and configure the remote OPC-UA server (in our case, if you are using the Docker implementation that you can find in the official GitHub repository, you can use `opc.tcp://opc-ua-server:26543`). Then, choose the AWS IoT Greengrass stream:



Add data source

Source type Info

OPC-UA
Connect the gateway to the server that uses the OPC-UA protocol to transfer industrial data.

Source name
A descriptive name lets you easily identify the source.

opc-ua-server local

The name must be unique in this gateway and have 1-256 characters.

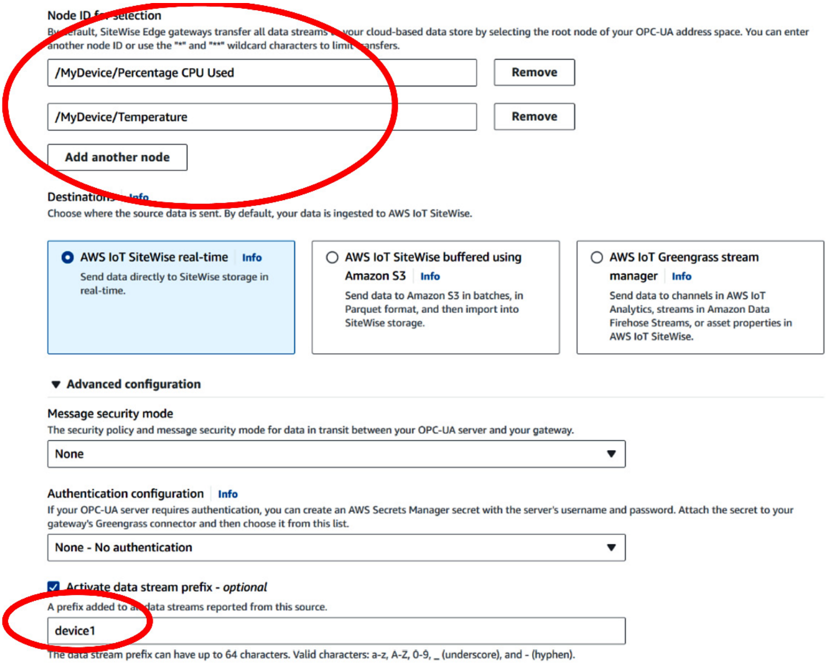
Local endpoint
The local endpoint for your OPC-UA server. The endpoint combines the internet protocol (IP) address or host, and (optional) a port number.

opc.tcp://opc-ua-server:26543

The local endpoint must have 1-256 characters. Valid characters: a-z, A-Z, 0-9, space, . (dot), : (colon), and / (slash), _ (underscore), and - (hyphen).

Figure 9.54 – Configuring the data source

1. Then, configure the nodes of the data source:



Node ID for selection

By default, SiteWise Edge gateways transfer all data streams to your cloud-based data store by selecting the root node of your OPC-UA address space. You can enter another node ID or use the "*" and "**" wildcard characters to limit transfers.

/MyDevice/Percentage CPU Used

/MyDevice/Temperature

Destination - Info

Choose where the source data is sent. By default, your data is ingested to AWS IoT SiteWise.

● AWS IoT SiteWise real-time Info
Send data directly to SiteWise storage in real-time.

○ AWS IoT SiteWise buffered using Amazon S3 Info
Send data to Amazon S3 in batches, in Parquet format, and then import into SiteWise storage.

○ AWS IoT Greengrass stream manager Info
Send data to channels in AWS IoT Analytics, streams in Amazon Data Firehose Streams, or asset properties in AWS IoT SiteWise.

▼ Advanced configuration

Message security mode

The security policy and message security mode for data in transit between your OPC-UA server and your gateway.

None

Authentication configuration Info

If your OPC-UA server requires authentication, you can create an AWS Secrets Manager secret with the server's username and password. Attach the secret to your gateway's Greengrass connector and then choose it from this list.

None - No authentication

☑ Activate data stream prefix - *optional*

A prefix added to all data streams reported from this source.

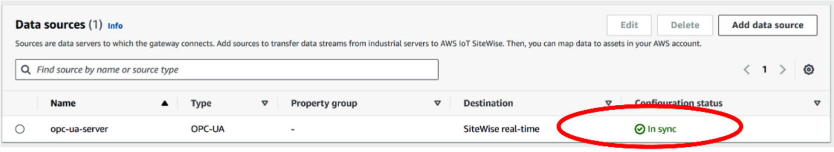
device1

The data stream prefix can have up to 64 characters. Valid characters: a-z, A-Z, 0-9, _ (underscore), and - (hyphen).

Figure 9.55 – Configuring the nodes of the data source

In our exercise, we are not configuring authentication and authorization mechanisms, but in general, it is always good practice to secure the resources with authentication and authorization enabled; we can provide authentication credentials on advanced configuration.

1. Wait a few seconds and check the status of the deployment:



Data sources (1) Info

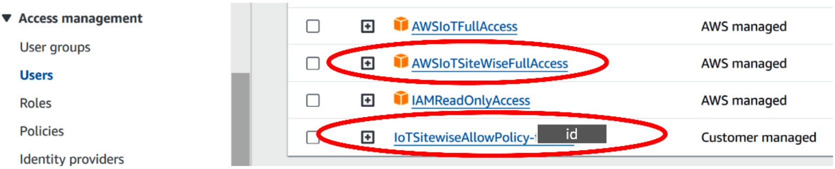
Sources are data servers to which the gateway connects. Add sources to transfer data streams from industrial servers to AWS IoT SiteWise. Then, you can map data to assets in your AWS account.

Edit
Delete
Add data source
Find source by name or source type
1

	Name	Type	Property group	Destination	Configuration status
○	opc-ua-server	OPC-UA	-	SiteWise real-time	✓ In sync

Figure 9.56 – Checking the status of the deployment

1. Update the IAM policy : We need to update the policies of the IAM user to allow SiteWise access. From the IAM service, click on the user and add the following policies:



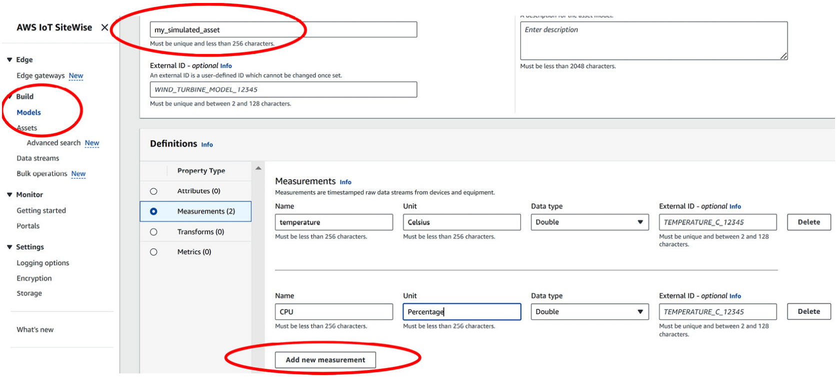
▼ Access management
User groups
Users
Roles
Policies
Identity providers

☐	⊞	AWSIoTFullAccess	AWS managed
☐	⊞	AWSIoTSiteWiseFullAccess	AWS managed
☐	⊞	IAMReadOnlyAccess	AWS managed
☐	⊞	IoTSitewiseAllowPolicy-[redacted]	Customer managed

Figure 9.57 – Adding the policies

1. Build an asset model and instance : SiteWise supports a very interesting feature; it can define a hierarchy of asset models. We have already seen this concept in previous chapters, where we called it the asset registry . Firstly, we need to define the asset model and then we can build the instance.

2. From Models | Assets , click on Create Asset Model and then set the name and add two measures: `CPU` and `temperature` . See the following figure:


AWS IoT SiteWise
Edge
Edge gateways New
Build
Models
Assets
Advanced search New
Data streams
Bulk operations New
Monitor
Getting started
Portals
Settings
Logging options
Encryption
Storage
What's new
Definitions Info
Property Type
○ Attributes (0)
● Measurements (2)
○ Transforms (0)
○ Metrics (0)
Delete
Delete
Add new measurement

my_simulated_asset

Must be unique and less than 256 characters.

External ID - *optional* Info

An external ID is a user-defined ID which cannot be changed once set.

WIND_TURBINE_MODEL_12345

Must be unique and between 2 and 128 characters.

Enter description

Must be less than 2048 characters.

Measurements Info

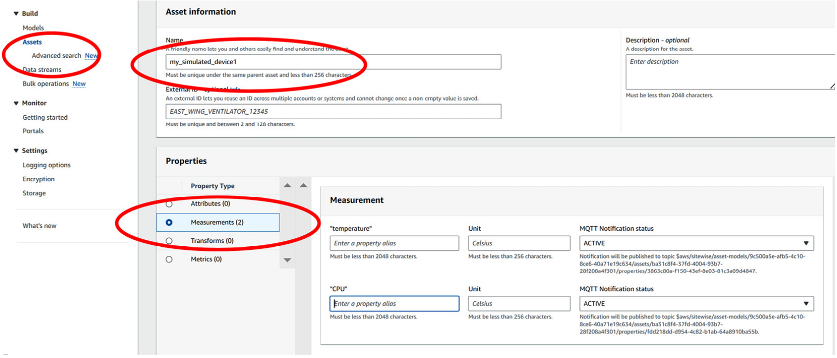
Measurements are timestamped raw data streams from devices and equipment.

Name	Unit	Data type	External ID - *optional* Info
temperature	Celsius	Double	*TEMPERATURE_C_12345*
CPU	Percentage	Double	*TEMPERATURE_C_12345*

Must be less than 256 characters. Must be unique and between 2 and 128 characters.

Figure 9.58 – Building the asset model

1. Then, create an asset instance. From Assets , click on Create asset and choose , provide a name, and choose the asset model just created:


Build
Models
Assets
Advanced search New
Data streams
Bulk operations New
Monitor
Getting started
Portals
Settings
Logging options
Encryption
Storage
What's new
Properties
Property Type
○ Attributes (0)
● Measurements (2)
○ Transforms (0)
○ Metrics (0)

Asset information

Name

A friendly name lets you and others easily find and understand the asset.

my_simulated_device1

Must be unique under the same parent asset and less than 256 characters.

External ID - optional Info

An external ID lets you reuse an ID across multiple accounts or systems and cannot change once a non-empty value is saved.

EAST_WING_VENTILATOR_12345

Must be unique and between 2 and 128 characters.

Description - *optional*

A description for the asset.

Enter description

Must be less than 2048 characters.

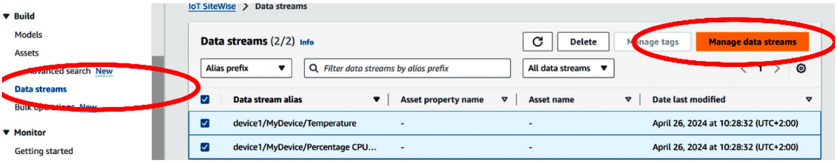
Measurement

	Unit	MQTT Notification status
"temperature" *Enter a property alias*	*Celsius*	ACTIVE Notification will be published to topic $aws/sitewise/asset-models/9c500a5e-afb5-4c10-8ce6-40a71e19c634/assets/ba31c8f4-37fd-4004-93b7-28f208a4f301/properties/3863c80a-f150-43ef-8e03-81c3a09d4847.
"CPU" *Enter a property alias*	*Celsius*	ACTIVE Notification will be published to topic $aws/sitewise/asset-models/9c500a5e-afb5-4c10-8ce6-40a71e19c634/assets/ba31c8f4-37fd-4004-93b7-28f208a4f301/properties/fdd218dd-d954-4c82-b1ab-64a8910ba55b.

Must be less than 2048 characters. Must be less than 256 characters.

Figure 9.59 – Building the asset instance

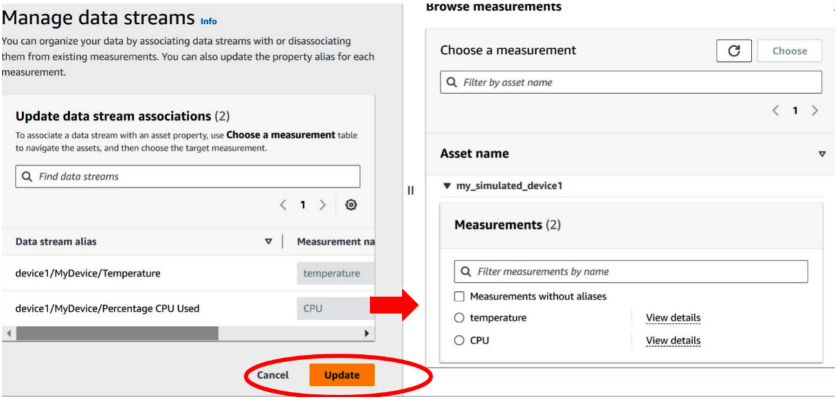
1. Link data source and asset model : Now, we can link the OPC data source with our asset model. From Data streams , click on the measures discovered by the OPC UA data source and click on Manage data streams . See the following figure:


Build
Models
Assets
Advanced search New
Data streams
Bulk operations New
Monitor
Getting started
IoT SiteWise > Data streams
Data streams (2/2) Info
Delete
Manage tags
Manage data streams
Alias prefix
Filter data streams by alias prefix
All data streams
1

☑	Data stream alias	Asset property name	Asset name	Date last modified
☑	device1/MyDevice/Temperature	-	-	April 26, 2024 at 10:28:32 (UTC+2:00)
☑	device1/MyDevice/Percentage CPU...	-	-	April 26, 2024 at 10:28:32 (UTC+2:00)

Figure 9.60 – Managing data streams

1. Finally, link the asset measure with the data source, as shown in the following figure:


Manage data streams Info
Update data stream associations (2)
Find data streams
1
Browse measurements
Choose a measurement
Choose
Filter by asset name
1
Asset name
my_simulated_device1
Measurements (2)
Filter measurements by name
Cancel
Update

You can organize your data by associating data streams with or disassociating them from existing measurements. You can also update the property alias for each measurement.

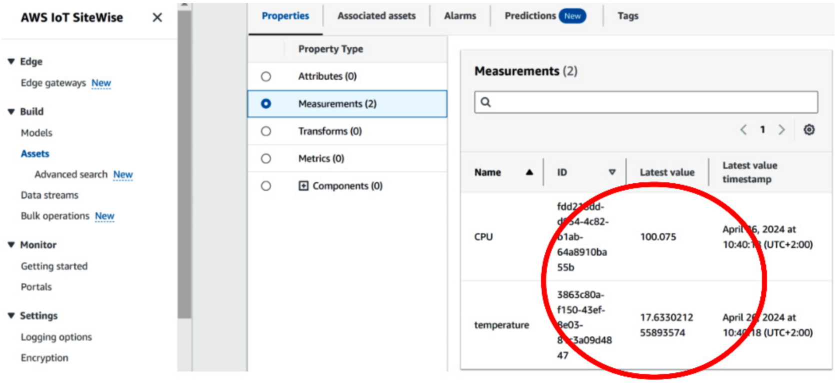
To associate a data stream with an asset property, use **Choose a measurement** table to navigate the assets, and then choose the target measurement.

Data stream alias	Measurement na
device1/MyDevice/Temperature	temperature
device1/MyDevice/Percentage CPU Used	CPU

☐ Measurements without aliases
○ temperature View details
○ CPU View details

Figure 9.61 – Managing data streams part 2

Well done! Now, you are ready to see data flowing into the cloud. From the asset in the Measurements section, you can see the new data:


AWS IoT SiteWise
Edge
Edge gateways New
Build
Models
Assets
Advanced search New
Data streams
Bulk operations New
Monitor
Getting started
Portals
Settings
Logging options
Encryption
Properties
Associated assets
Alarms
Predictions New
Tags
Property Type
Attributes (0)
Measurements (2)
Transforms (0)
Metrics (0)
Components (0)

Measurements (2)

Name	ID	Latest value	Latest value timestamp
CPU	fdd2[illegible]dd-d[illegible]54-4c82-[illegible]1ab-64a8910ba55b	100.075	April [illegible]6, 2024 at 10:40:[illegible] (UTC+2:00)
temperature	3863c80a-f150-43ef-[illegible]e03-8[illegible]3a09d4847	17.633021255893574	April 2[illegible], 2024 at 10:4[illegible]18 (UTC+2:00)

Figure 9.62 – Data published on the SiteWise cloud

In this section, we studied SiteWise and configured it to read data from the OPC-UA server and send it to the AWS cloud.

About SiteWise

SiteWise is a good component but, currently, it still seems a bit immature; however, it has all the necessary concepts to support IIoT including other protocols such as Siemens and Modbus. At the same time, we can develop custom measures based on a formula (e.g., the maximum of a measure).

Moreover, SiteWise allows us to develop a local web portal (a minimal human-machine interface) to monitor data on-site (see the *Monitor | Portals* section in the left menu in Figure 9 .62). SiteWise is very powerful and it is the IIoT solution proposed by AWS. Now that we have our cloud data, we want to process it; in the next chapter, we will build our data flow and our analytical EMAs. We have already done something similar in *Chapter 8* , so we will do the same exercise again with AWS.

Summary

In this chapter, we explored the most common functionalities used to deliver an IoT solution with AWS. We understood the main components and the benefits.

We developed our first IoT system with AWS; we saw some types of edge offered by AWS: Greengrass, the AWS SDK, and SiteWise. We also used a custom device (Node-RED) to show how AWS IoT is universally compatible with MQTT. In the next chapter, we will finalize our exercise by implementing an end-to-end data flow.

We would like to highlight that we are using the same flow proposed in *Chapter 7* and sometimes readjusting it by reusing the same technology or changing the components with similar ones.

10

Implementing an Industrial IOT Data Flow with AWS

In the previous chapter, we implemented the edge-to-cloud AWS Industrial IoT (IIoT) data flow. We looked at some important topics, including storage, time series, microservices, protocols, data stream processing, and batch processing. This chapter explores the AWS solutions used to implement an operational data flow.

In particular, we will cover the following topics:

- Storing data on Amazon Timestream or S3
- AWS IoT Analytics
- AWS IoT Events
- Grafana

Technical requirements

To follow along with this chapter, the following prerequisites should be downloaded and installed:

- Git : https://git-scm.com/downloads
- Node.js 20+ : https://nodejs.org
- Python 3.9+ : https://www.python.org/downloads
- Docker : https://www.docker.com/products/docker-desktop/

The code for this chapter is available at the official repository at https://github.com/PacktPublishing/Hands-On-Industrial-Internet-of-Things-Second-Edition .

Architecture of the exercise

To discover AWS data flow capabilities, we will finalize our exercise based on IoT Core, AWS Lambda, MQTT, the AWS edge SDK, IoT Analytics, and Grafana. The following figure illustrates the architecture introduced in *Chapter 9* . In the next sections, we will implement the data flow (the highlighted right part):

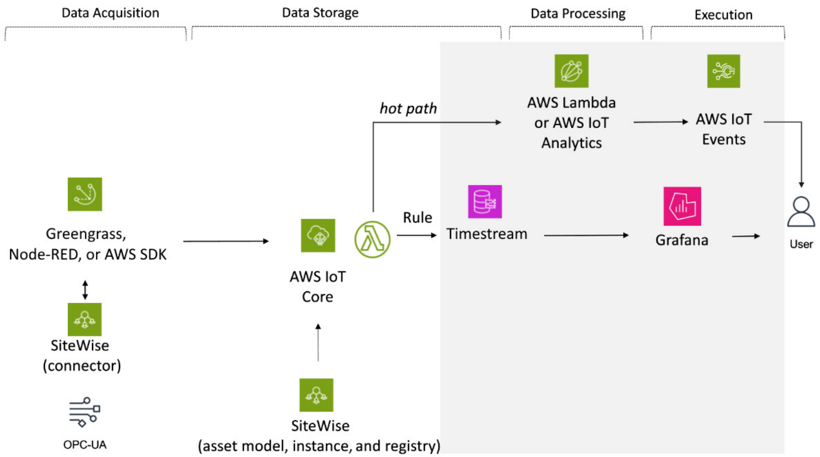

Data Acquisition
Data Storage
Data Processing
Execution
hot path
AWS Lambda or AWS IoT Analytics
AWS IoT Events
Greengrass, Node-RED, or AWS SDK
AWS IoT Core
Rule
Timestream
Grafana
User
SiteWise (connector)
OPC-UA
SiteWise (asset model, instance, and registry)

Figure 10.1 – Architecture of our exercise

AWS IoT Analytics

In *Chapter 9* , we leveraged AWS IoT and the AWS edge or AWS SDK to send data to the cloud. Once the devices are connected and the time-series data is sent to the cloud, IIoT data should be analyzed, processed, and visualized. AWS provides six different mechanisms to process data:

- Serverless Lambda functions

- Greengrass, which can be used to deploy Lambda functions on-premises
- IoT Analytics
- Athena
- SageMaker

SageMaker and Athena

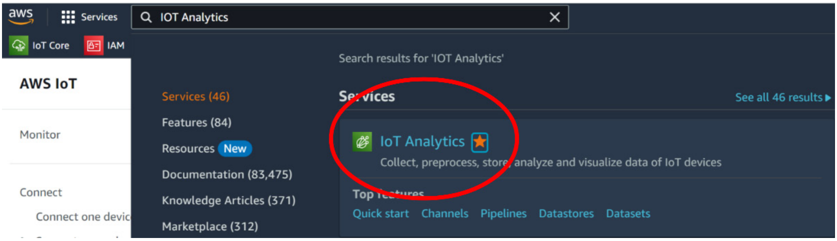
AWS SageMaker is a general-purpose machine learning and deep learning framework used to develop advanced analytics. SageMaker works very well with S3, so we need to create a batch to export data continuously from Timestream to S3. This operation is very easy to configure using AWS Data Pipeline. AWS Athena is a powerful SQL query language for business intelligence that is easy to apply to NoSQL's context.

IoT Analytics

IoT Analytics is another analytical framework for IoT data transformation and enrichment. IoT

Analytics is a single service that can process the full chain of our data, from data processing to data enrichment.

To work with IoT Analytics, we need to build the channels, the pipelines, and finally, the data stores. These are, respectively, the input, the transformation, and the output. To work with AWS IoT Analytics, connect the browser to the IoT console and search for IoT Analytics .


aws
Services
IOT Analytics
IoT Core
IAM
AWS IoT
Monitor
Connect
Connect one devic
Search results for 'IOT Analytics'
Services (46)
Features (84)
Resources New
Documentation (83,475)
Knowledge Articles (371)
Marketplace (312)
Services
See all 46 results
IoT Analytics
Top features
Quick start
Channels
Pipelines
Datastores
Datasets

Collect, preprocess, store, analyze and visualize data of IoT devices

Figure 10.2 – Starting AWS IoT Analytics

Now, we want to reuse one of our just-developed edges to send data to AWS IoT and process it directly on AWS IoT Analytics. For simplicity, we are going to reuse the first device developed with AWS SDK.

Implementing a hot data path with AWS IoT Analytics

Now, we want to perform a simple threshold-based anomaly detection algorithm using IoT Analytics. During this exercise, we will do the following:

- Connect IoT Core to stream data into IoT Analytics
- Implement local data storage to save our data
- Apply a simple rule to enrich the dataset
- Visualize and process the dataset

Let's begin!

Part 1 – connecting to IoT Core : We need to build our channel, and we want to connect IoT Analytics to our IoT Core topics:

1. From the menu on the left, click on Channels , and then click on Create channel :

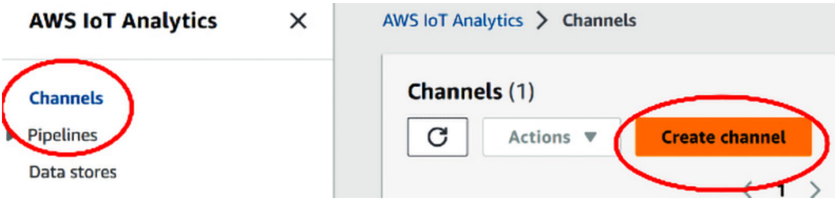
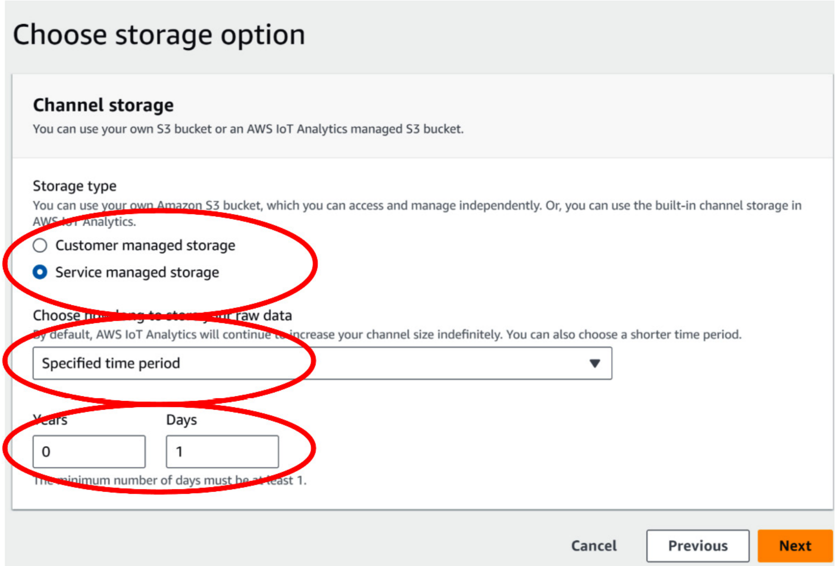

AWS IoT Analytics
Channels
Pipelines
Data stores
AWS IoT Analytics > Channels
Channels (1)
Actions ▼
Create channel

Figure 10.3 – Create channel

1. Next, provide a name for the channel (e.g., `aws_iot`) and then click the Next button.

2. In the storage section, specify Service managed storage and a time period of `1` day to not incur high costs:



Choose storage option

Channel storage

You can use your own S3 bucket or an AWS IoT Analytics managed S3 bucket.

Storage type

You can use your own Amazon S3 bucket, which you can access and manage independently. Or, you can use the built-in channel storage in AWS IoT Analytics.

○ Customer managed storage

● Service managed storage

Choose how long to store your raw data

By default, AWS IoT Analytics will continue to increase your channel size indefinitely. You can also choose a shorter time period.

Specified time period ▼

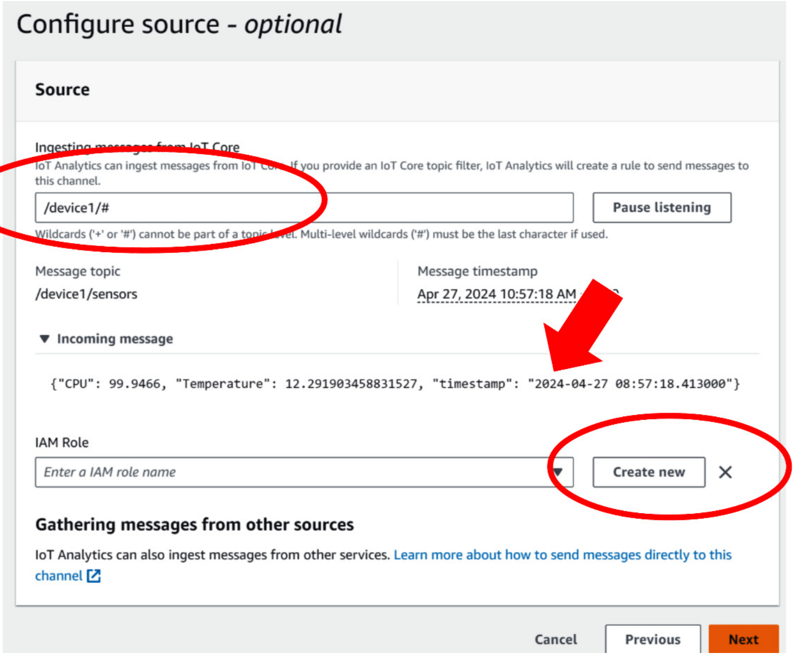
Years: 0

Days: 1

The minimum number of days must be at least 1.

Figure 10.4 – Specifying storage

1. On the IoT Core topic filter field, we can subscribe to the `/device1/# MQTT` topic and then click on See messages to see the messages flowing into IoT Analytics:



Configure source - *optional*

Source

Ingesting messages from IoT Core

IoT Analytics can ingest messages from IoT Core. If you provide an IoT Core topic filter, IoT Analytics will create a rule to send messages to this channel.

/device1/#

Wildcards ('+' or '#') cannot be part of a topic level. Multi-level wildcards ('#') must be the last character if used.

Message topic: /device1/sensors

Message timestamp: Apr 27, 2024 10:57:18 AM

▼ Incoming message

```
{"CPU": 99.9466, "Temperature": 12.291903458831527, "timestamp": "2024-04-27 08:57:18.413000"}
```

IAM Role

Enter a IAM role name

Gathering messages from other sources

IoT Analytics can also ingest messages from other services. Learn more about how to send messages directly to this channel

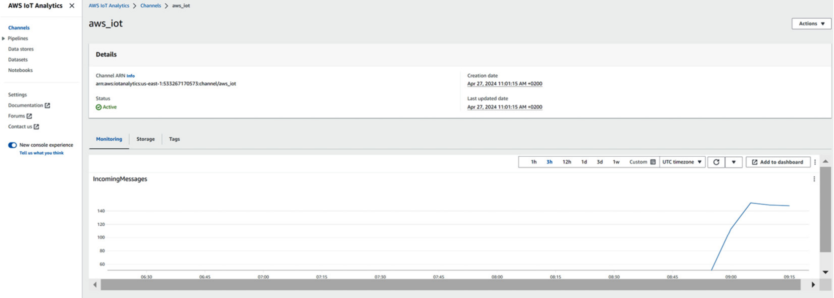
Figure 10.5 – Connect IoT Core sources

1. Then, create the new role, click Next , review the channel, and click Create channel .

> ## *How does it work?*

> *AWS IoT Analytics will create a topic rule for us on IoT Core. You can see it in* ***IoT Core*** *|* ***Message Routing*** *|* ***Rules*** *. Do not forget to delete rules if you want to remove channels. Moreover, AWS IoT Analytics creates a role to push from IoT Core to the channel just created. If you want to add another pipeline, you need to create another policy or edit the IAM role just created from the IAM service.*

1. To check the data, click on Monitoring , and after a few minutes, you will see the number of messages sent by IoT Core:


AWS IoT Analytics
Channels
Pipelines
Data stores
Datasets
Notebooks
Settings
Documentation
Forums
Contact us
New console experience
Tell us what you think
AWS IoT Analytics > Channels > aws_iot
aws_iot
Actions
Details
Monitoring
Storage
Tags
1h
3h
12h
1d
3d
1w
Custom
UTC timezone
Add to dashboard
IncomingMessages
140
120
100
80
60
06:30
06:45
07:00
07:15
07:30
07:45
08:00
08:15
08:30
08:45
09:00
09:15

Channel ARN Info
arn:aws:iotanalytics:us-east-1:533267170573:channel/aws_iot

Status
Active

Creation date
Apr 27, 2024 11:01:15 AM +0200

Last updated date
Apr 27, 2024 11:01:15 AM +0200

Figure 10.6 – Validate channel

Part 2 – building the data store and the pipeline :

The first step is to create data storage for our results:

1. From Data stores , create a new store, as outlined here:

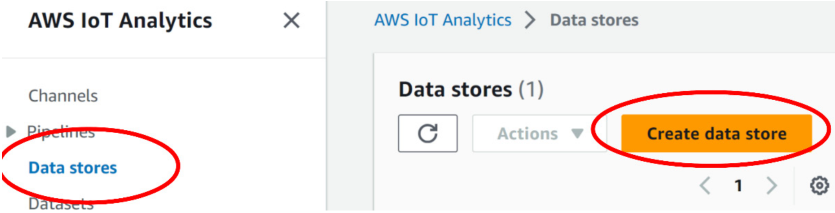
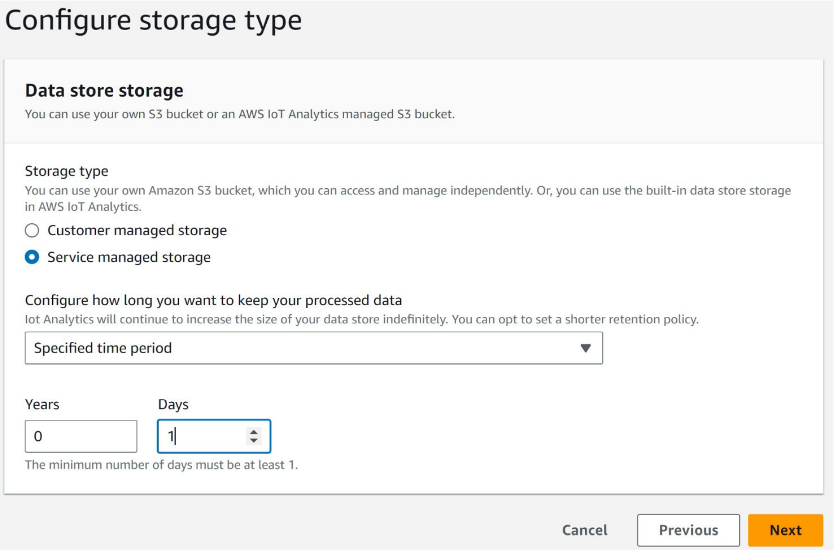

AWS IoT Analytics
Channels
Pipelines
Data stores
Datasets
AWS IoT Analytics > Data stores
Data stores (1)
Actions
Create data store
1

Figure 10.7 – Creating a new data store

1. Then, provide a name and set the storage type to Service managed storage with retention of about `1` day. Finally, accept all steps and click on Create data storage .



Configure storage type

Data store storage

You can use your own S3 bucket or an AWS IoT Analytics managed S3 bucket.

Storage type

You can use your own Amazon S3 bucket, which you can access and manage independently. Or, you can use the built-in data store storage in AWS IoT Analytics.

○ Customer managed storage

● Service managed storage

Configure how long you want to keep your processed data

Iot Analytics will continue to increase the size of your data store indefinitely. You can opt to set a shorter retention policy.

Specified time period

Years: 0

Days: 1

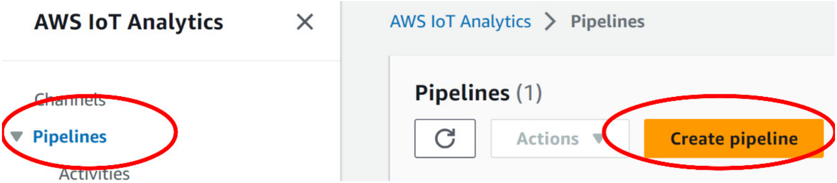
The minimum number of days must be at least 1.

Cancel
Previous
Next

Figure 10.8 – Configure storage type

Part 3 – creating a pipeline from a channel :

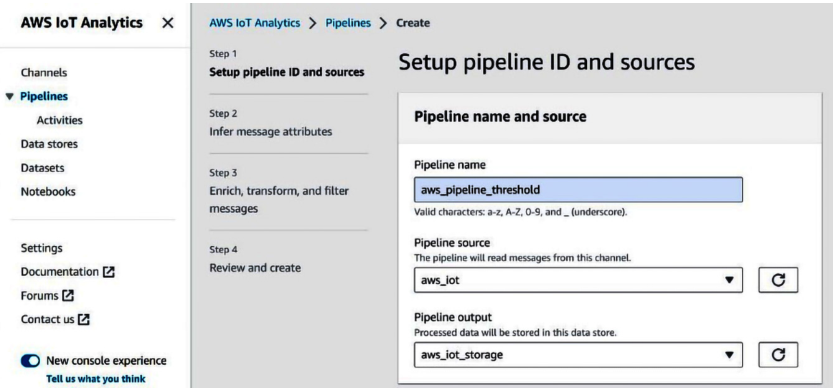
1. To build our transformation activity, we can create the pipeline from the channel, as outlined here:



AWS IoT Analytics
Channels
Pipelines
Activities
AWS IoT Analytics > Pipelines
Pipelines (1)
Actions
Create pipeline

Figure 10.9 – Create pipeline

1. We need to provide a name, such as **`aws_pipeline_threshold`**, and then define the source and output, as follows:


AWS IoT Analytics
Channels
Pipelines
Activities
Data stores
Datasets
Notebooks
Settings
Documentation
Forums
Contact us
New console experience
Tell us what you think
AWS IoT Analytics > Pipelines > Create
Step 1
Setup pipeline ID and sources
Step 2
Infer message attributes
Step 3
Enrich, transform, and filter messages
Step 4
Review and create

Setup pipeline ID and sources

Pipeline name and source

Pipeline name: aws_pipeline_threshold

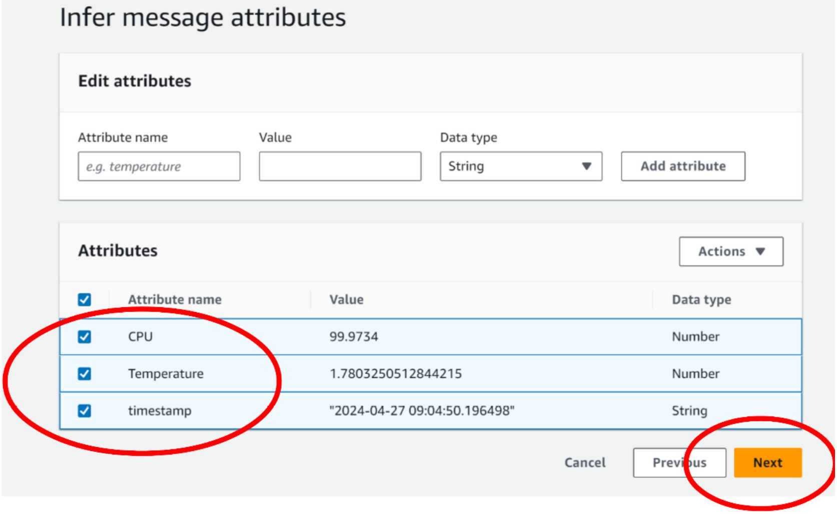
Valid characters: a-z, A-Z, 0-9, and _ (underscore).

Pipeline source: The pipeline will read messages from this channel. aws_iot

Pipeline output: Processed data will be stored in this data store. aws_iot_storage

Figure 10.10 – Define the source and destination of a pipeline

1. We can easily build attribute calculation and enrichment with just a few clicks. In the next step, accept all attributes, as outlined here:

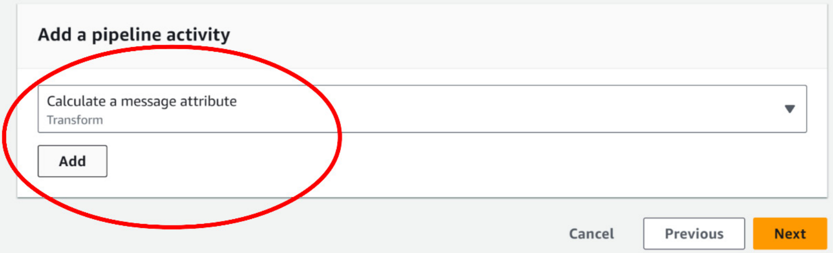

Infer message attributes
Edit attributes
Attribute name
e.g. temperature
Value
Data type
String
Add attribute
Attributes
Actions
Cancel
Previous
Next

☑	Attribute name	Value	Data type
☑	CPU	99.9734	Number
☑	Temperature	1.7803250512844215	Number
☑	timestamp	"2024-04-27 09:04:50.196498"	String

Figure 10.11 – Choose measures of the data source

1. The next step is the most important; add a new activity, in this case,

Calculate a message attribute :

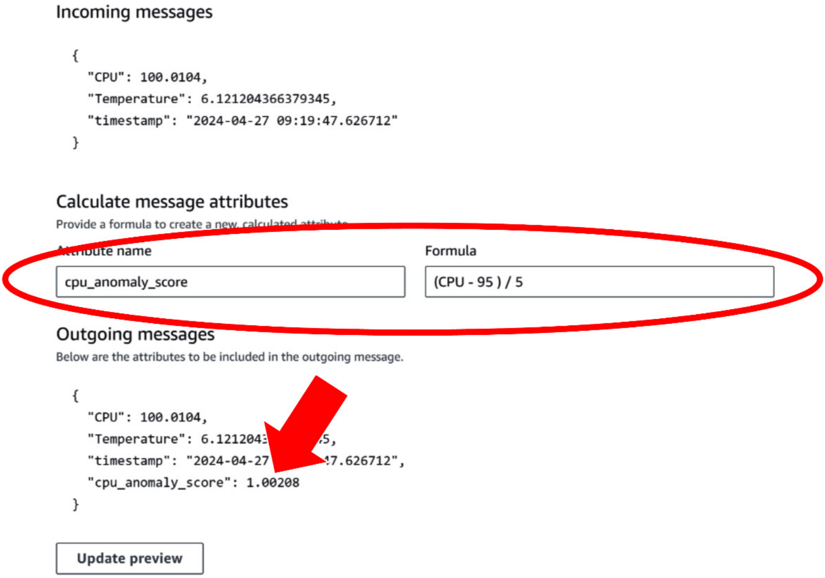


Add a pipeline activity

Calculate a message attribute
Transform

Figure 10.12 – Add Calculate a message attribute

1. Then, define an anomaly score so that if the CPU is greater than `95`, `cpu_anomaly_score` is greater than `1`, as follows:



Incoming messages

```
{
  "CPU": 100.0104,
  "Temperature": 6.121204366379345,
  "timestamp": "2024-04-27 09:19:47.626712"
}
```

Calculate message attributes

Provide a formula to create a new calculated attribute

Attribute name: cpu_anomaly_score

Formula: (CPU - 95) / 5

Outgoing messages

Below are the attributes to be included in the outgoing message.

```
{
  "CPU": 100.0104,
  "Temperature": 6.1212043      5,
  "timestamp": "2024-04-27      7.626712",
  "cpu_anomaly_score": 1.00208
}
```

Figure 10.13 – Messages

1. Click Next and accept all the next steps.

Tip

> *Alternatively, you can build your custom function using Lambda. See https://docs.aws.amazon.com/iotanalytics/latest/userguide/pipeline-activities-lambda.html .*

Part 4 – preparing the dataset : To work with the pipeline we just created, we need to ask the IoT Analytics framework to create a dataset periodically:

1. From the menu on the left, click on Analysis , then Datasets , and then Create dataset :

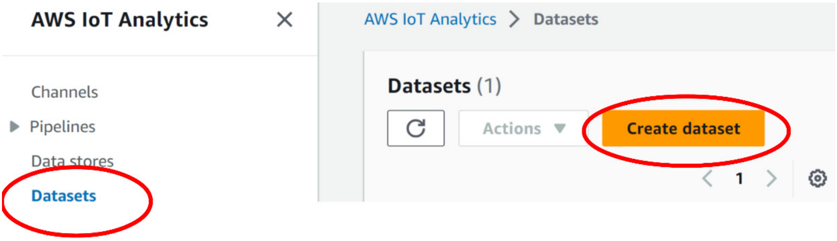
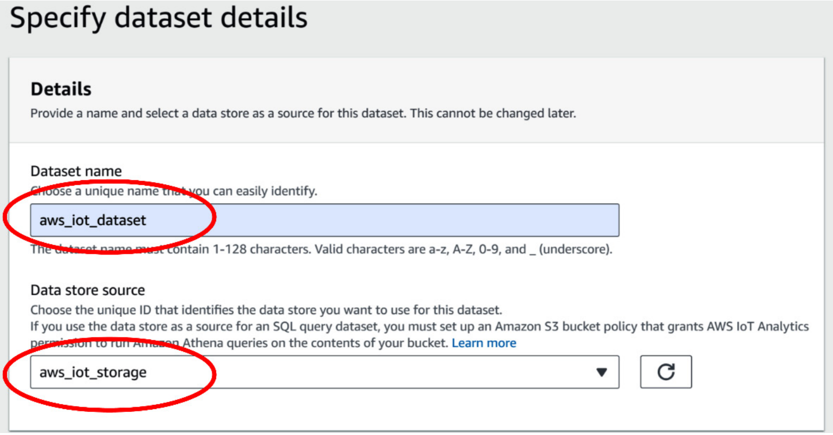

AWS IoT Analytics
Channels
Pipelines
Data stores
Datasets
AWS IoT Analytics > Datasets
Datasets (1)
Actions
Create dataset
1

Figure 10.14 – Creating a new dataset

1. In the next step, choose Create SQL and provide a name. Enter the name **`aws_iot_dataset`** and select **`aws_iot_storage`** , which we created previously, as the data store source, as shown in the following screenshot:



Specify dataset details

Details

Provide a name and select a data store as a source for this dataset. This cannot be changed later.

Dataset name

Choose a unique name that you can easily identify.

aws_iot_dataset

The dataset name must contain 1-128 characters. Valid characters are a-z, A-Z, 0-9, and _ (underscore).

Data store source

Choose the unique ID that identifies the data store you want to use for this dataset.

If you use the data store as a source for an SQL query dataset, you must set up an Amazon S3 bucket policy that grants AWS IoT Analytics permission to run Amazon Athena queries on the contents of your bucket. Learn more

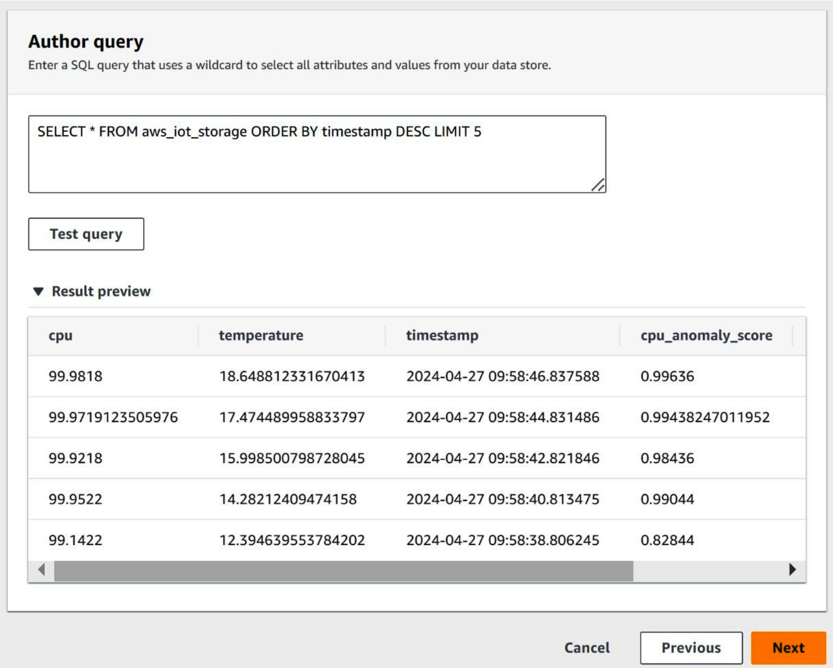
aws_iot_storage

Figure 10.15 – Define dataset name and storage

1. In the next step, we can define our SQL:

```
SELECT * FROM aws_iot_storage ORDER
BY timestamp DESC LIMIT 100
```

See the following figure, which shows the expected results for just five elements:



Author query

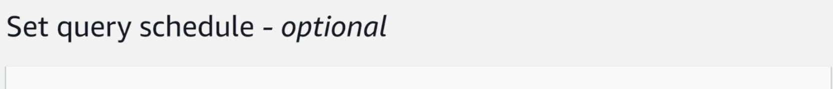
Enter a SQL query that uses a wildcard to select all attributes and values from your data store.

```
SELECT * FROM aws_iot_storage ORDER BY timestamp DESC LIMIT 5
```

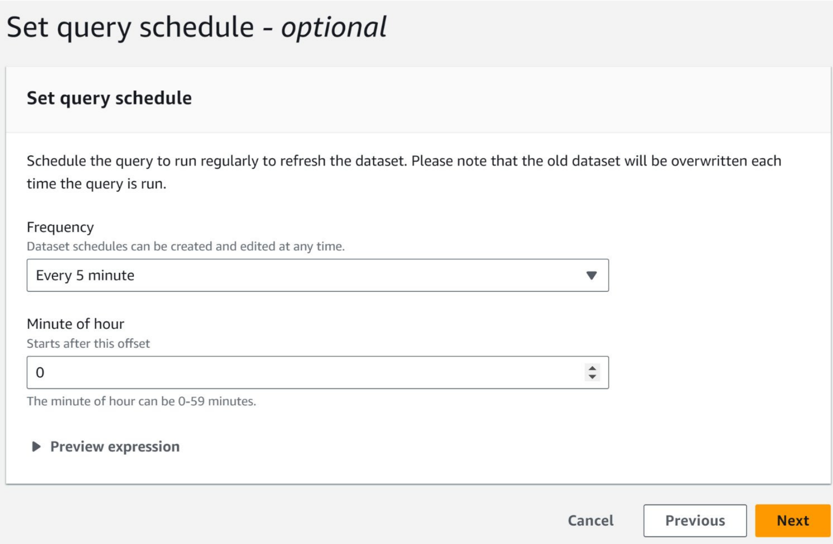
cpu	temperature	timestamp	cpu_anomaly_score
99.9818	18.648812331670413	2024-04-27 09:58:46.837588	0.99636
99.9719123505976	17.474489958833797	2024-04-27 09:58:44.831486	0.99438247011952
99.9218	15.998500798728045	2024-04-27 09:58:42.821846	0.98436
99.9522	14.28212409474158	2024-04-27 09:58:40.813475	0.99044
99.1422	12.394639553784202	2024-04-27 09:58:38.806245	0.82844

Figure 10.16 – Custom SQL

1. In the next step, provide how long you want to keep the results. As usual, you can define `1` day. Click Next and set your scheduling interval to Every 5 minute :



Set query schedule - *optional*



Set query schedule - *optional*

Set query schedule

Schedule the query to run regularly to refresh the dataset. Please note that the old dataset will be overwritten each time the query is run.

Frequency

Dataset schedules can be created and edited at any time.

Every 5 minute

Minute of hour

Starts after this offset

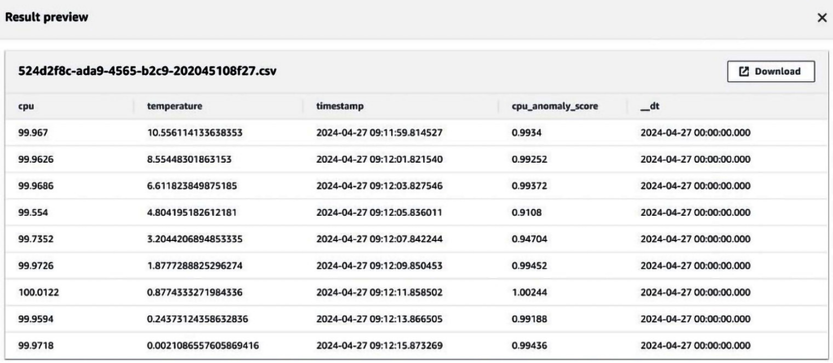
0

The minute of hour can be 0-59 minutes.

Figure 10.17 – Schedule every five minutes

1. Finally, we can build our dataset by clicking on the Next button. After creation, click on the Run query now option.

The IoT Analytics framework is a low-latency integration platform for fast transformation and prototyping. You can see the dataset by clicking on the last version and then Content . The expected results should show `cpu_anomaly_score` :



524d2f8c-ada9-4565-b2c9-202045108f27.csv

cpu	temperature	timestamp	cpu_anomaly_score	__dt
99.967	10.556114133638353	2024-04-27 09:11:59.814527	0.9934	2024-04-27 00:00:00.000
99.9626	8.55448301863153	2024-04-27 09:12:01.821540	0.99252	2024-04-27 00:00:00.000
99.9686	6.611823849875185	2024-04-27 09:12:03.827546	0.99372	2024-04-27 00:00:00.000
99.554	4.804195182612181	2024-04-27 09:12:05.836011	0.9108	2024-04-27 00:00:00.000
99.7352	3.2044206894853335	2024-04-27 09:12:07.842244	0.94704	2024-04-27 00:00:00.000
99.9726	1.8777288825296274	2024-04-27 09:12:09.850453	0.99452	2024-04-27 00:00:00.000
100.0122	0.8774333271984336	2024-04-27 09:12:11.858502	1.00244	2024-04-27 00:00:00.000
99.9594	0.24373124358632836	2024-04-27 09:12:13.866505	0.99188	2024-04-27 00:00:00.000
99.9718	0.0021086557605869416	2024-04-27 09:12:15.873269	0.99436	2024-04-27 00:00:00.000

Figure 10.18 – CPU anomaly score

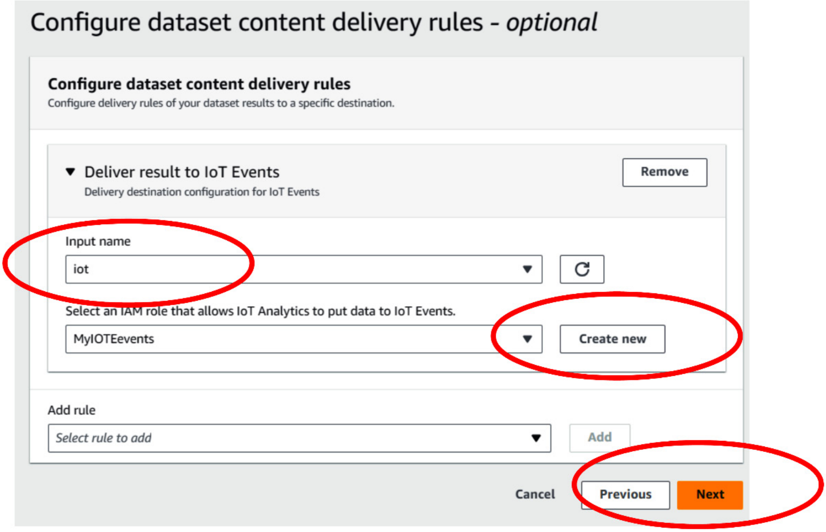
About IoT Analytics

IoT Analytics allows us to build a fast pipeline to process online data. When the dataset is ready, a data scientist can start the analysis by clicking on Notebook . Alternatively, you can decide to send data to IoT Events to raise alarms or to S3 to visualize data using Grafana or QuickSight.

IoT Events

AWS IoT Events is a component able to manage events and build a state machine model. AWS IoT Events can be configured to work with SiteWise or IoT Analytics. To connect IoT Analytics with IoT

Events, you need to build a new dataset (or reuse the last one), define a new rule to send the output to IoT Events, choose the input name, create a new role, and finally click on Next , as shown in the following figure:



Configure dataset content delivery rules - *optional*

Configure dataset content delivery rules

Configure delivery rules of your dataset results to a specific destination.

▼ Deliver result to IoT Events

Delivery destination configuration for IoT Events

Input name: iot

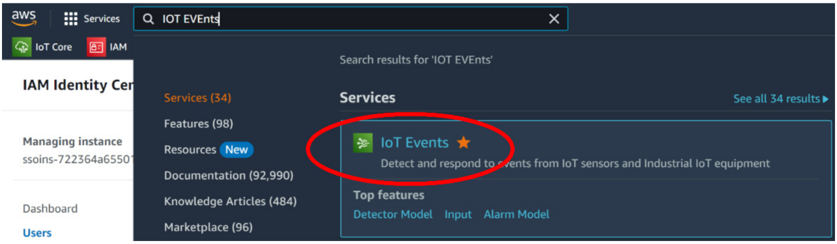
Select an IAM role that allows IoT Analytics to put data to IoT Events.: MyIOTEevents

Add rule: *Select rule to add*

Figure 10.19 – Deliver results to IoT Events

Now, we can configure IoT Events to define an alerting system; in our case, we want to generate an alarm when the CPU is greater than `95` . Follow these steps:

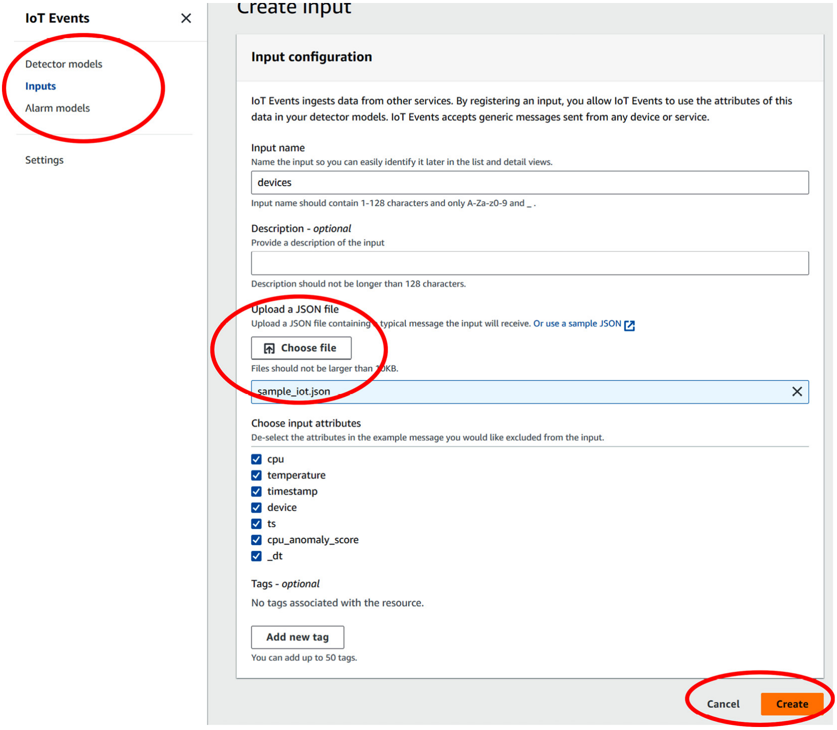
1. From Services , search `IoT Events` and click on IoT Events :


aws
Services
IOT EVEnts
IoT Core
IAM
IAM Identity Cer
Managing instance
ssoins-722364a6550
Dashboard
Users
Search results for 'IOT EVEnts'
Services (34)
Features (98)
Resources New
Documentation (92,990)
Knowledge Articles (484)
Marketplace (96)
Services
See all 34 results
IoT Events
Top features
Detector Model
Input
Alarm Model

Detect and respond to events from IoT sensors and Industrial IoT equipment

Figure 10.20 – Locate the IoT Events service

1. From the left menu, we need to add a new input; then, we can define the name, provide a simple JSON sample of input, and click on Create , as shown here:


IoT Events
Detector models
Inputs
Alarm models
Settings
Create input
Choose file
Add new tag
Cancel
Create

Input configuration

IoT Events ingests data from other services. By registering an input, you allow IoT Events to use the attributes of this data in your detector models. IoT Events accepts generic messages sent from any device or service.

Input name
Name the input so you can easily identify it later in the list and detail views.
devices
Input name should contain 1-128 characters and only A-Za-z0-9 and _ .

Description - *optional*
Provide a description of the input
Description should not be longer than 128 characters.

Upload a JSON file
Upload a JSON file containing a typical message the input will receive. Or use a sample JSON
Files should not be larger than 10KB.
sample_iot.json

Choose input attributes
De-select the attributes in the example message you would like excluded from the input.
☑ cpu
☑ temperature
☑ timestamp
☑ device
☑ ts
☑ cpu_anomaly_score
☑ _dt

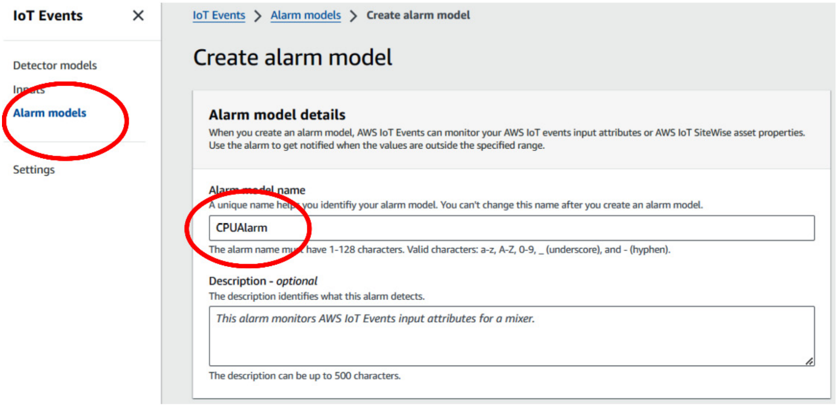
Tags - *optional*
No tags associated with the resource.
You can add up to 50 tags.

Figure 10.21 – Create an input for the alert model

IoT Events is not able to infer the structure of the input; so, as an input sample, we can use a flat JSON with all the keys. Here is an example:

```
{"cpu": 99.9616,
"temperature": 18.685912493786464,
"timestamp": "2024-04-27
13:28:13.282933",
"device": "device1",
"ts": "2024-04-27 15:33:06.885 UTC" ,
"cpu_anomaly_score" : 0.9}
```

1. Now, we are ready to define our alert model. From the left menu, we can now create an alarm model, and we can call it **`CPUAlarm`**:



Create alarm model

Alarm model details

When you create an alarm model, AWS IoT Events can monitor your AWS IoT events input attributes or AWS IoT SiteWise asset properties. Use the alarm to get notified when the values are outside the specified range.

Alarm model name

A unique name helps you identify your alarm model. You can't change this name after you create an alarm model.

CPUAlarm

The alarm name must have 1-128 characters. Valid characters: a-z, A-Z, 0-9, _ (underscore), and - (hyphen).

Description - *optional*

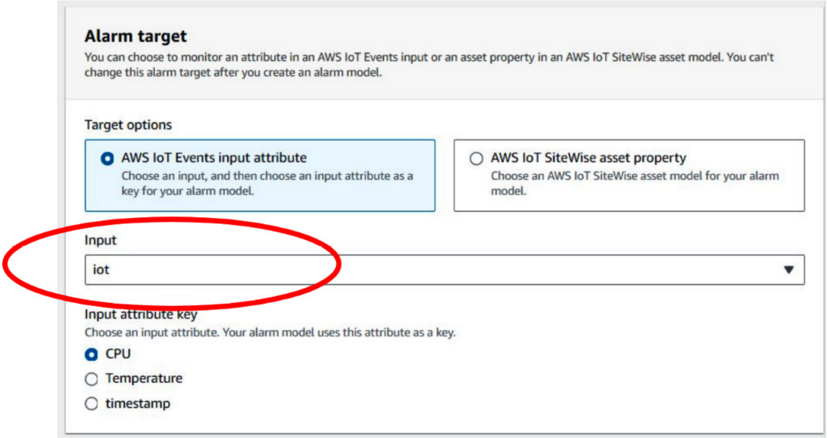
The description identifies what this alarm detects.

This alarm monitors AWS IoT Events input attributes for a mixer.

The description can be up to 500 characters.

Figure 10.22 – Create an alert model

We reuse the input just created before:



Alarm target

You can choose to monitor an attribute in an AWS IoT Events input or an asset property in an AWS IoT SiteWise asset model. You can't change this alarm target after you create an alarm model.

Target options

● AWS IoT Events input attribute
Choose an input, and then choose an input attribute as a key for your alarm model.

○ AWS IoT SiteWise asset property
Choose an AWS IoT SiteWise asset model for your alarm model.

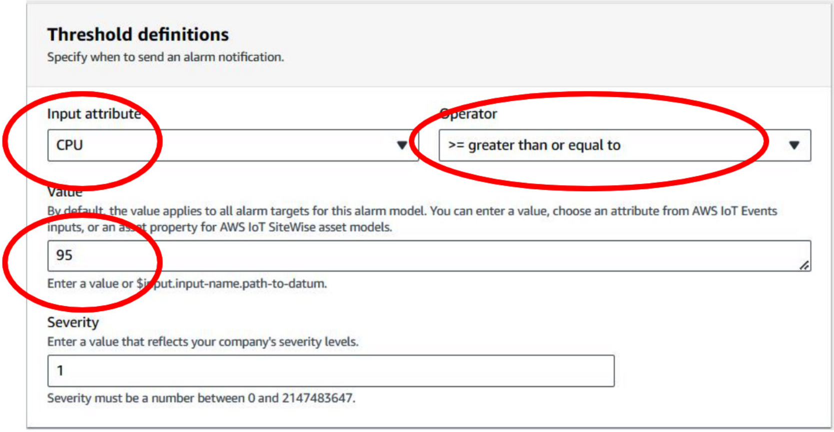
Input: iot

Input attribute key
Choose an input attribute. Your alarm model uses this attribute as a key.

● CPU
○ Temperature
○ timestamp

Figure 10.23 – Select the input of the alert model

1. Finally, we define our rule; for example, we raise a text message when the CPU is greater than `95`. See the following figure:



Threshold definitions

Specify when to send an alarm notification.

Input attribute: CPU

Operator: >= greater than or equal to

Value
By default, the value applies to all alarm targets for this alarm model. You can enter a value, choose an attribute from AWS IoT Events inputs, or an asset property for AWS IoT SiteWise asset models.

95

Enter a value or $input.input-name.path-to-datum.

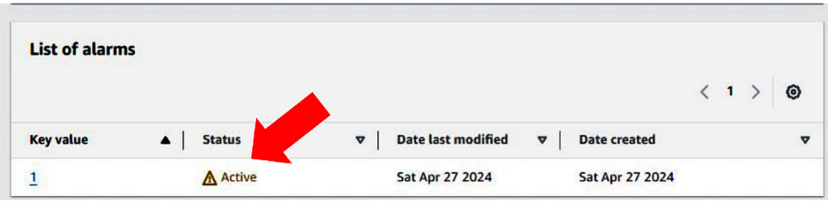
Severity
Enter a value that reflects your company's severity levels.

1

Severity must be a number between 0 and 2147483647.

Figure 10.24 – Define the rule of asset model

We can now create our alert model. After a few minutes, we can see our first raised alarm:



List of alarms

Key value	Status	Date last modified	Date created
1	⚠ Active	Sat Apr 27 2024	Sat Apr 27 2024

Figure 10.25 – See the alarm raised by the asset model

Leveraging IoT Events, we have developed an excursion monitoring analytic (EMA) in a few clicks. We will develop more complex analytics in *Chapter 13* , after studying the concepts of machine learning, artificial intelligence, and digital twins.

More about IoT Events

IoT Events is an interesting service to develop our excursion management system (EMS). IoT Events also supports a state machine model called a detector model ; through a detector model, we can define a state machine and define a transition mechanism among states.

Moreover, IoT Events also supports SiteWise natively; indeed, IoT Events can use the asset model as an example of input and connect directly to SiteWise.

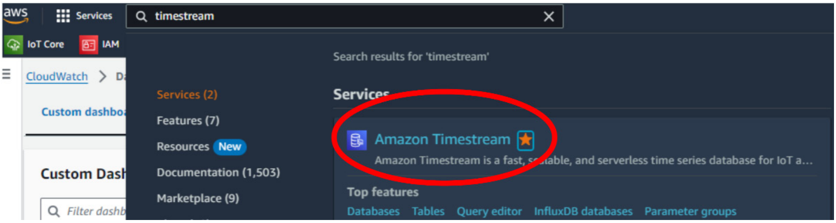
In the previous section, we developed our online analytics, but sometimes, we need to store our raw data to be visualized or processed later by batch algorithms; in the next section, we will discover Timestream and other storage databases.

Storing data

In a typical IIoT scenario, we want to process incoming data, but also store this data in a cloud storage system. We can receive from the device unstructured data, such as images, sounds, or logs, as well as events and sensor data. In our proposed IIoT architecture, we want to store sensor data, such as time-series data, but it is easy to extend these concepts to unstructured data as well. AWS supports time-series data with Amazon Timestream and Managed InfluxDB natively. For our exercise, we will use Timestream .

Timestream is a key-value-based NoSQL store service for high scalability. In this exercise, we are going to finalize our exercise by sending data from the edge to Timestream through IoT Core. In the next section, we will visualize the live data from Grafana.

Firstly, we need to search **`Timestream`** from the Services console:

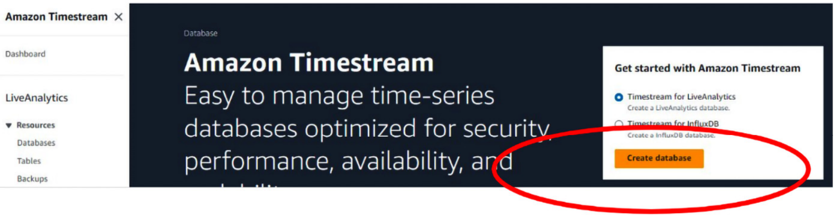

aws
Services
timestream
IoT Core
IAM
CloudWatch > D
Custom dashbo
Custom Dash
Filter dashb
Search results for 'timestream'
Services (2)
Features (7)
Resources New
Documentation (1,503)
Marketplace (9)
Services
Amazon Timestream
Top features
Databases
Tables
Query editor
InfluxDB databases
Parameter groups

Amazon Timestream is a fast, scalable, and serverless time series database for IoT a...

Figure 10.26 – Locate Timestream

Now, we need to create a database, create a table, and inject data from IoT Core to Timestream. Let's start:

1. To set up a Timestream database, from the Timestream console, click on Create database :



Amazon Timestream
Dashboard
LiveAnalytics
Resources
Databases
Tables
Backups
Database
Amazon Timestream
Get started with Amazon Timestream
Create database

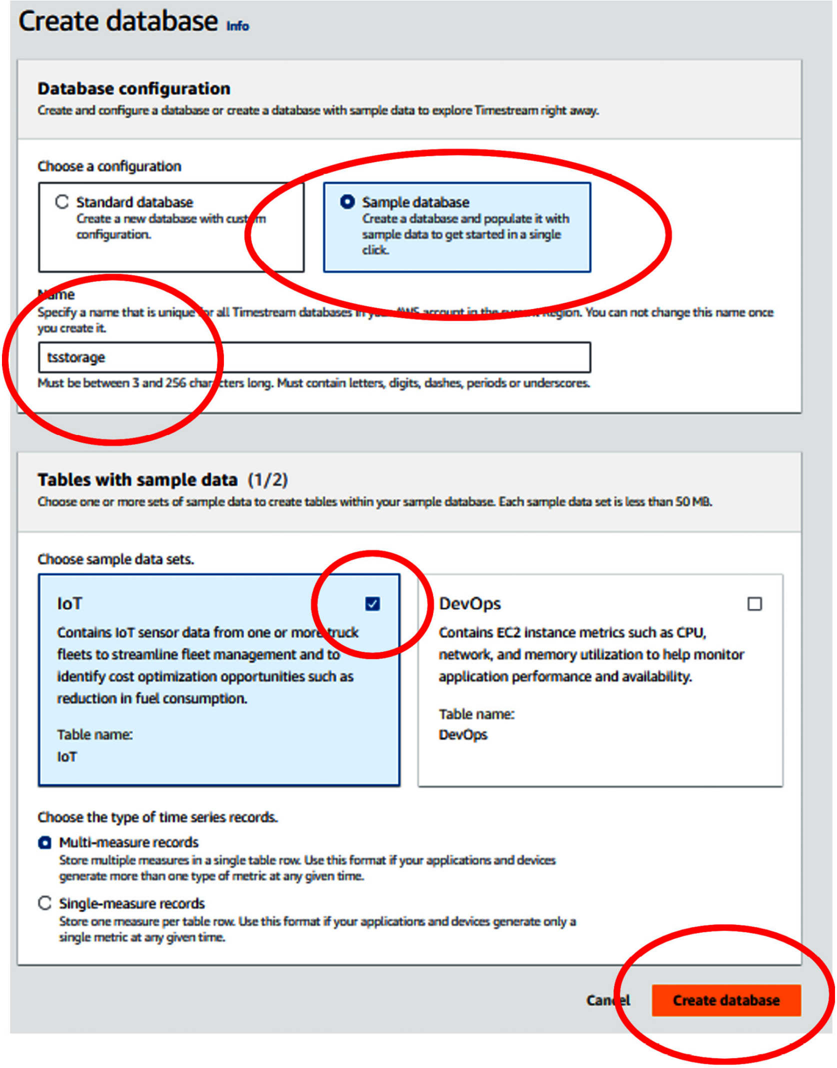
Easy to manage time-series databases optimized for security, performance, availability, and

● Timestream for LiveAnalytics
Create a LiveAnalytics database.

○ Timestream for InfluxDB
Create a InfluxDB database.

Figure 10.27 – Create database

1. Now, we can provide the name. We will use a sample IoT database for multi-measure records. Timestream supports both tables with multiple columns and key-value type tables; for this exercise, we prefer the multi-column ones. Configure the database as shown here:



Create database

Database configuration

Create and configure a database or create a database with sample data to explore Timestream right away.

Choose a configuration

○ Standard database
Create a new database with custom configuration.

● Sample database
Create a database and populate it with sample data to get started in a single click.

Name
Specify a name that is unique for all Timestream databases in your AWS account in the current Region. You can not change this name once you create it.

tsstorage

Must be between 3 and 256 characters long. Must contain letters, digits, dashes, periods or underscores.

Tables with sample data (1/2)

Choose one or more sets of sample data to create tables within your sample database. Each sample data set is less than 50 MB.

Choose sample data sets.

IoT ☑
Contains IoT sensor data from one or more truck fleets to streamline fleet management and to identify cost optimization opportunities such as reduction in fuel consumption.
Table name:
IoT

DevOps ☐
Contains EC2 instance metrics such as CPU, network, and memory utilization to help monitor application performance and availability.
Table name:
DevOps

Choose the type of time series records.

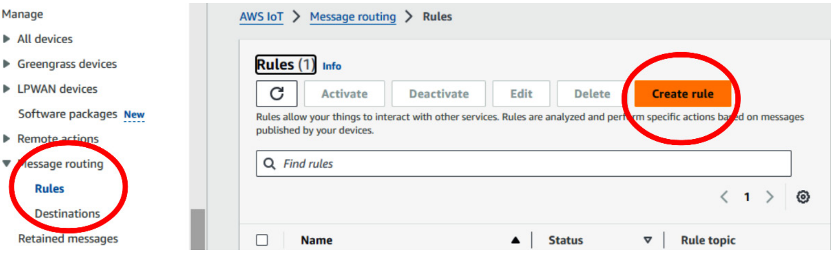
● Multi-measure records
Store multiple measures in a single table row. Use this format if your applications and devices generate more than one type of metric at any given time.

○ Single-measure records
Store one measure per table row. Use this format if your applications and devices generate only a single metric at any given time.

Figure 10.28 – Create multi-column DB

Timestream will create a database with a table called IoTMulti .

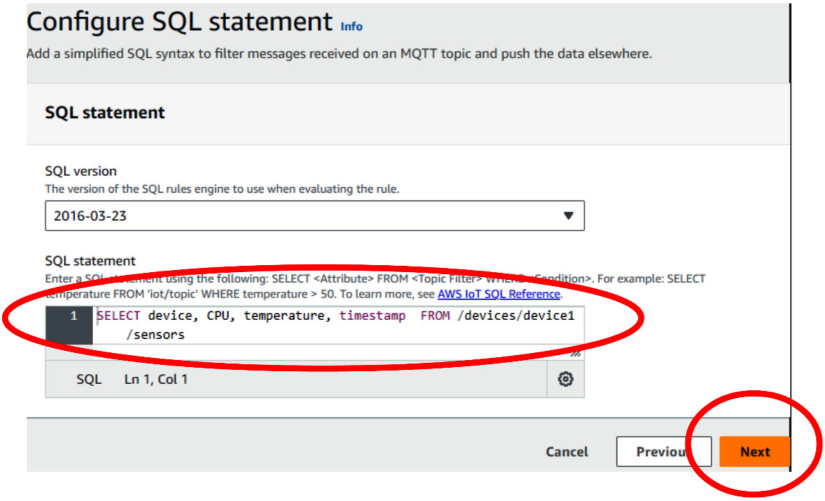
1. To connect the edge to Timestream, go to IoT Core | Message Routing | Rules | Create rule :


Manage
All devices
Greengrass devices
LPWAN devices
Software packages New
Remote actions
Message routing
Rules
Destinations
Retained messages
AWS IoT > Message routing > Rules
Rules (1) Info
Activate
Deactivate
Edit
Delete
Create rule
Find rules
1
Name
Status
Rule topic

Rules allow your things to interact with other services. Rules are analyzed and perform specific actions based on messages published by your devices.

Figure 10.29 – Create a rule to connect IoT Core to Timestream

1. Define a name (e.g., `Device2Timestream`). Click Next and in the SQL statement, provide a query to read from the `/devices/device1/sensors` topic, as shown here:


Configure SQL statement Info
SQL statement
SQL version
SQL statement
SQL
Ln 1, Col 1
Cancel
Previous
Next

Add a simplified SQL syntax to filter messages received on an MQTT topic and push the data elsewhere.

The version of the SQL rules engine to use when evaluating the rule.

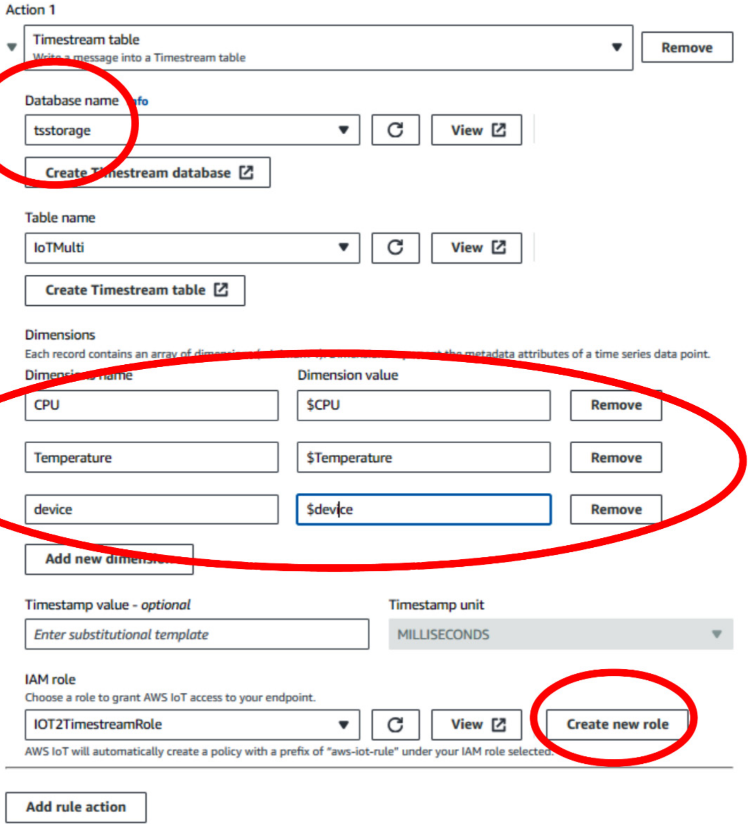
2016-03-23

Enter a SQL statement using the following: SELECT <Attribute> FROM <Topic Filter> WHERE <Condition>. For example: SELECT temperature FROM 'iot/topic' WHERE temperature > 50. To learn more, see AWS IoT SQL Reference.

```
SELECT device, CPU, temperature, timestamp  FROM /devices/device1
    /sensors
```

Figure 10.30 – Define an SQL statement to read from the IoT Core topic

1. Finally, map each input to the right column, as shown in the following figure:



Action 1

Timestream table
Write a message into a Timestream table

Database name: tsstorage

Table name: IoTMulti

Dimensions
Each record contains an array of dimensions (at least 1). Dimensions represent the metadata attributes of a time series data point.

Dimension name	Dimension value
CPU	$CPU
Temperature	$Temperature
device	$device

Timestamp value - *optional*: *Enter substitutional template*

Timestamp unit: MILLISECONDS

IAM role
Choose a role to grant AWS IoT access to your endpoint.
IOT2TimestreamRole
AWS IoT will automatically create a policy with a prefix of "aws-iot-rule" under your IAM role selected.

Remove
Info
View
Create Timestream database
View
Create Timestream table
Remove
Remove
Remove
Add new dimension
View
Create new role
Add rule action

Figure 10.31 – Map source to destination

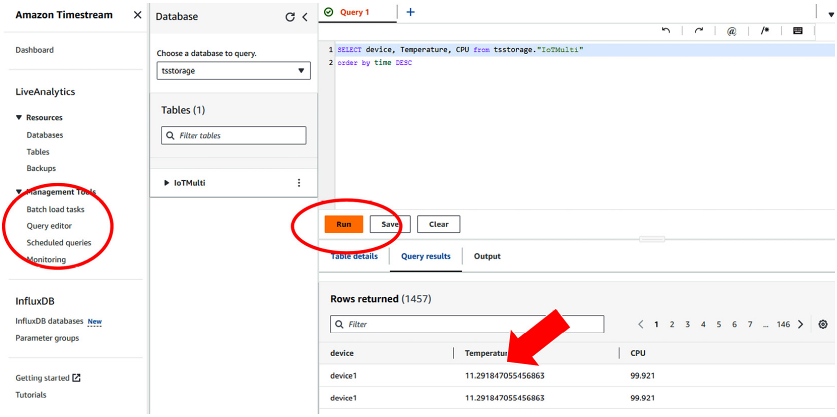
Do not forget to create a new role for this activity.

Now, you can execute a simple query from the query builder:

```
select device, CPU, Temperature from
tsstorage."IoTMulti" order by time
DESC
```

We can see the data sent by the edge and stored on

Timestream:


Amazon Timestream
Dashboard
LiveAnalytics
Resources
Databases
Tables
Backups
Management Tools
Batch load tasks
Query editor
Scheduled queries
Monitoring
InfluxDB
InfluxDB databases New
Parameter groups
Getting started
Tutorials
Database
Choose a database to query.
tsstorage
Tables (1)
Filter tables
IoTMulti
Query 1
Run
Save
Clear
Table details
Query results
Output
Filter
1 2 3 4 5 6 7 ... 146

```
SELECT device, Temperature, CPU from tsstorage."IoTMulti"
order by time DESC
```

Rows returned (1457)

device	Temperature	CPU
device1	11.291847055456863	99.921
device1	11.291847055456863	99.921

Figure 10.32 – Visualizing data

Timestream supports several functions and operators; for example, the following query uses the **GROUP BY** operator and the **AVG** and **BIN** functions to interpolate the CPU every two minutes:

```
SELECT device,
BIN(time, 120s) AS binned_timestamp,
AVG(CAST(CPU AS double)) AS
avg_cpu_utilization
FROM tsstorage."IoTMulti"
GROUP BY device, BIN(time, 120s)
order by binned_timestamp DESC
```

Asset model

> *In our exercise, we have mapped each measure to a Timestream column. We suggest this approach only if the asset model doesn't change too much among the types of machines. If we plan to monitor mixed types of assets (e.g., WindTurbine, CNC, Solar, etc.), we suggest using the* `asset-id-key-value` *structure to avoid complex mapping from IoT Core to Timestream.*

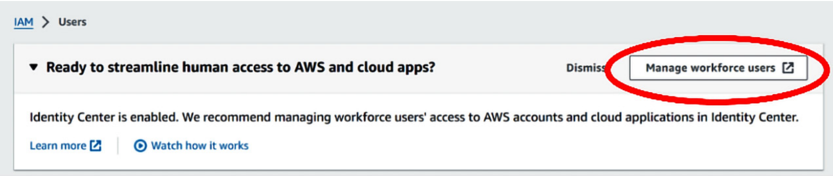
Finally, we have our data saved to be processed or simply viewed; in the next paragraph, we will see how.

Visualization

Alert, report, and measure visualizations are the final steps of our workflow: finally, we need to provide insights to users. The fastest way to build a powerful user interface for a user is to reuse the Grafana framework. We have already implemented a Grafana service in *Chapter 8* ; in the next section, we will start using the Amazon Managed Grafana service.

Workforce users

Before working with Grafana, we must fulfill a prerequisite: we need to add the final operator (a user) to our security system. From IAM , click on Users and then Manage workforce users :

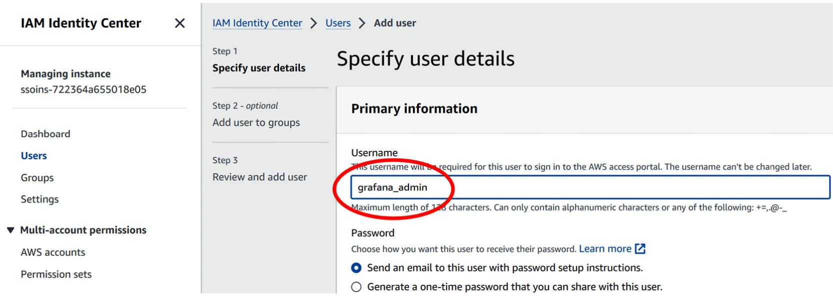

IAM > Users
Dismiss
Manage workforce users
Learn more
Watch how it works

Ready to streamline human access to AWS and cloud apps?

Identity Center is enabled. We recommend managing workforce users' access to AWS accounts and cloud applications in Identity Center.

Figure 10.33 – Create a workforce

Then, create a new user and call them **`grafana_admin`** :


IAM Identity Center
Dashboard
Users
Groups
Settings
Multi-account permissions
AWS accounts
Permission sets
IAM Identity Center > Users > Add user
Step 1
Specify user details
Step 2 - optional
Add user to groups
Step 3
Review and add user
Specify user details
Primary information
Learn more

Managing instance
ssoins-722364a655018e05

Username

This username will be required for this user to sign in to the AWS access portal. The username can't be changed later.

grafana_admin

Maximum length of 128 characters. Can only contain alphanumeric characters or any of the following: +=,.@-_

Password

Choose how you want this user to receive their password.

● Send an email to this user with password setup instructions.

○ Generate a one-time password that you can share with this user.

Figure 10.34 – Create a Grafana admin user

We can build several users for different roles (Editor, Admin, Viewer). Provide a valid email address and AWS IAM will send an email to validate the user, provide a password, and configure MFA.

Grafana

To work with Grafana, search **`Grafana`** in Services and click on Amazon Grafana :

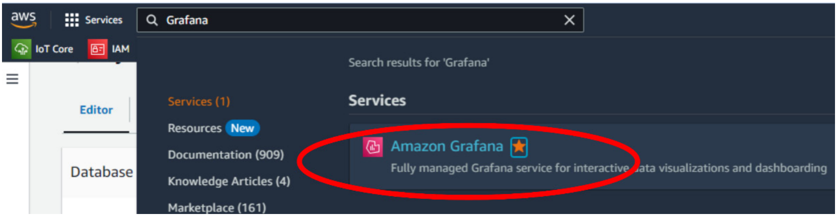
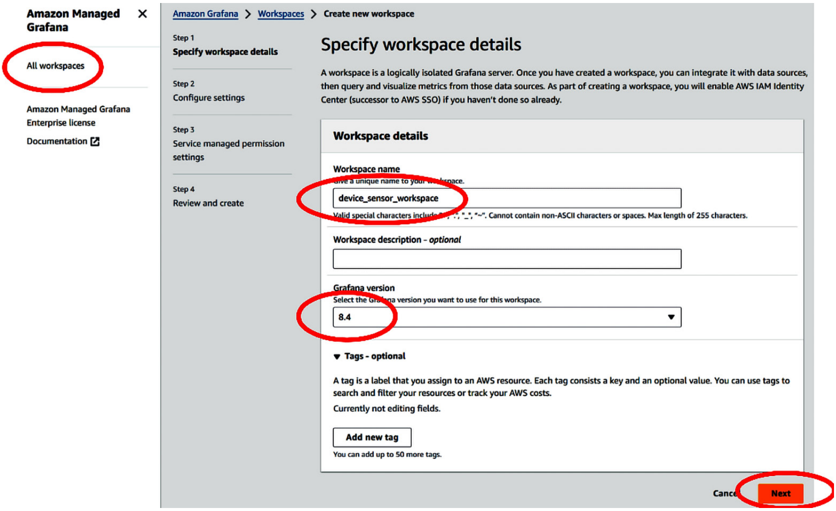

aws
Services
Grafana
IoT Core
IAM
Editor
Database
Search results for 'Grafana'
Services (1)
Resources New
Documentation (909)
Knowledge Articles (4)
Marketplace (161)
Services
Amazon Grafana
Fully managed Grafana service for interactive ata visualizations and dashboarding

Figure 10.35 – Locate Grafana

Now, we need to create an instance of Grafana (workspace), attach the previous user, configure the data source, and build our dashboard. Let's do so by following these steps:

1. To create a workspace, from the left menu of Grafana, select All workspaces and Create new workspace . Choose Grafana 8+ and click on Next :


Amazon Managed Grafana
All workspaces
Amazon Managed Grafana Enterprise license
Documentation
Amazon Grafana > Workspaces > Create new workspace
Step 1
Specify workspace details
Step 2
Configure settings
Step 3
Service managed permission settings
Step 4
Review and create

Specify workspace details

A workspace is a logically isolated Grafana server. Once you have created a workspace, you can integrate it with data sources, then query and visualize metrics from those data sources. As part of creating a workspace, you will enable AWS IAM Identity Center (successor to AWS SSO) if you haven't done so already.

Workspace details

Workspace name
Give a unique name to your workspace.
device_sensor_workspace
Valid special characters include [illegible] ".", "_", "~". Cannot contain non-ASCII characters or spaces. Max length of 255 characters.

Workspace description - *optional*

Grafana version
Select the Grafana version you want to use for this workspace.
8.4

▼ Tags - optional

A tag is a label that you assign to an AWS resource. Each tag consists a key and an optional value. You can use tags to search and filter your resources or track your AWS costs.

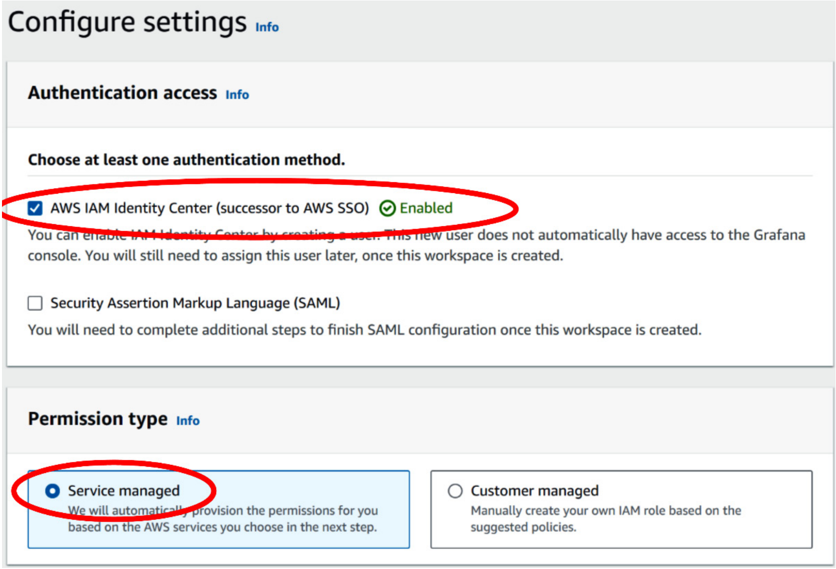
Currently not editing fields.

You can add up to 50 more tags.

Cancel
Next

Figure 10.36 – Create a workspace

1. In the next step, choose the AWS IAM Identity Center method and Service managed :



Configure settings

Authentication access

Choose at least one authentication method.

☑ AWS IAM Identity Center (successor to AWS SSO) ✓ Enabled

You can enable IAM Identity Center by creating a user. This new user does not automatically have access to the Grafana console. You will still need to assign this user later, once this workspace is created.

☐ Security Assertion Markup Language (SAML)

You will need to complete additional steps to finish SAML configuration once this workspace is created.

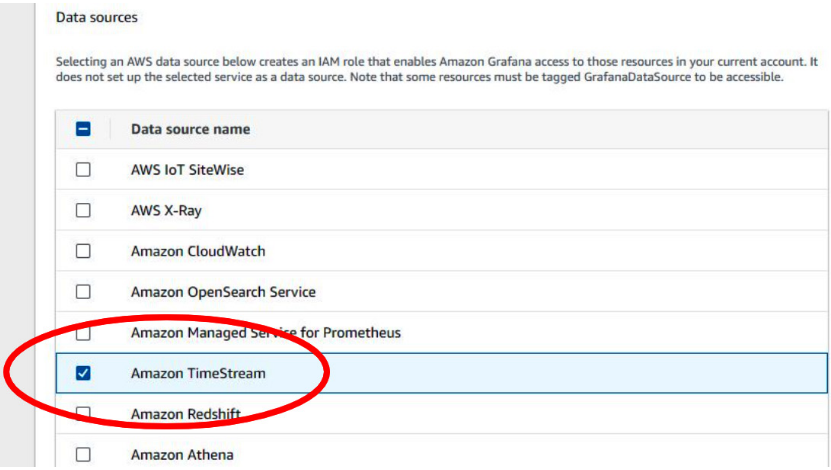
Permission type

● Service managed
We will automatically provision the permissions for you based on the AWS services you choose in the next step.

○ Customer managed
Manually create your own IAM role based on the suggested policies.

Figure 10.37 – Configure settings for Grafana

Now, we are ready to attach the Timestream service and allow Grafana to explore the list of our Timestream databases and tables:



Data sources

Selecting an AWS data source below creates an IAM role that enables Amazon Grafana access to those resources in your current account. It does not set up the selected service as a data source. Note that some resources must be tagged GrafanaDataSource to be accessible.

⊟	Data source name
☐	AWS IoT SiteWise
☐	AWS X-Ray
☐	Amazon CloudWatch
☐	Amazon OpenSearch Service
☐	Amazon Managed Service for Prometheus
☑	Amazon TimeStream
☐	Amazon Redshift
☐	Amazon Athena

Figure 10.38 – Connect Grafana to Timestream

AWS will take a few minutes to build the Grafana workspace.

1. To allow users to connect to Grafana, click on the workspace just created, and on the Authentication tab, click on Configure Users and Groups and assign the Grafana Administrator user to Grafana:

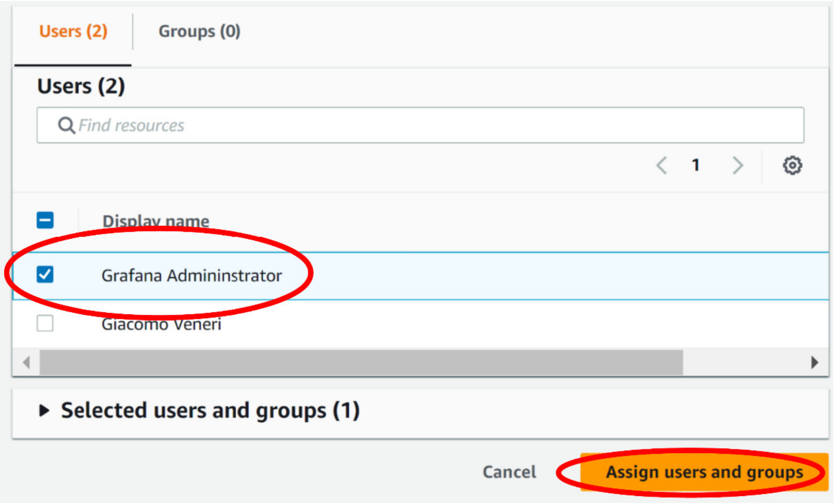
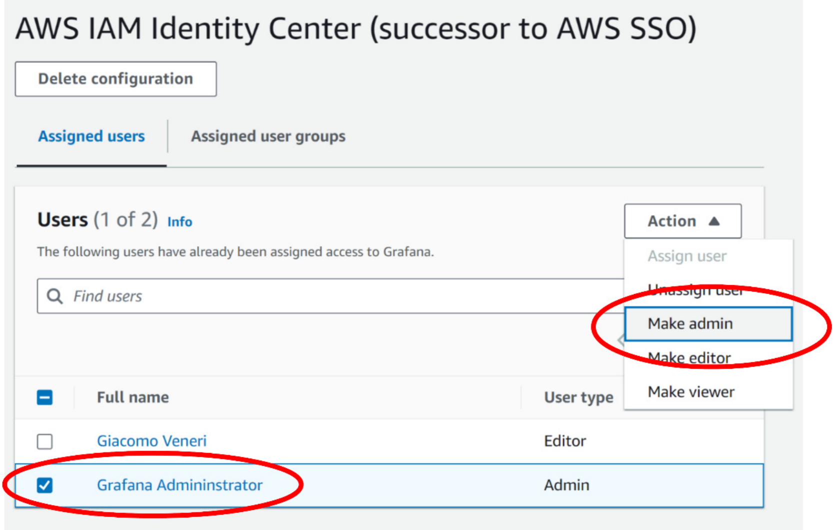

Users (2)
Groups (0)
Users (2)
Find resources
1
Display name
Grafana Admininstrator
Giacomo Veneri
Selected users and groups (1)
Cancel
Assign users and groups

Figure 10.39 – Allow users to access Grafana

1. Finally, make the user the administrator, as shown here:


AWS IAM Identity Center (successor to AWS SSO)
Delete configuration
Assigned users
Assigned user groups
Users (1 of 2) Info
Action
Assign user
Unassign user
Make admin
Make editor
Make viewer
Find users
Full name
User type

The following users have already been assigned access to Grafana.

	Full name	User type
☐	Giacomo Veneri	Editor
☑	Grafana Admininstrator	Admin

Figure 10.40 – Make the user administrator

You can add multiple users with different roles, for instance, I added myself as Editor, but the user must be valid; in other words, the user must have accepted the invitation and configured a password and MFA.

1. Next, to configure a data source and Grafana and connect our database, click on the Grafana workspace URL and you will be redirected to the public Grafana site. You must log in with the Grafana Administrator user created during the previous section. From the left menu of Grafana, click on Data sources and Add data source :

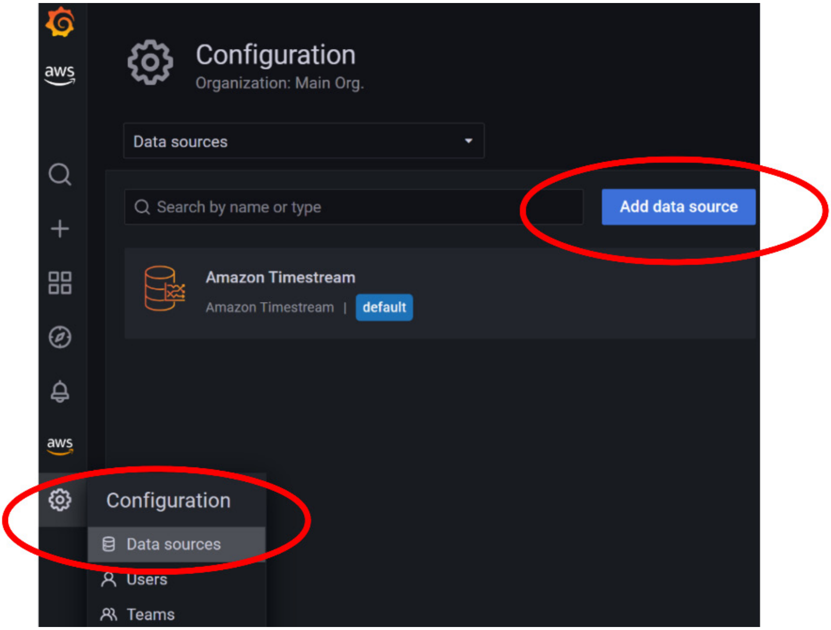
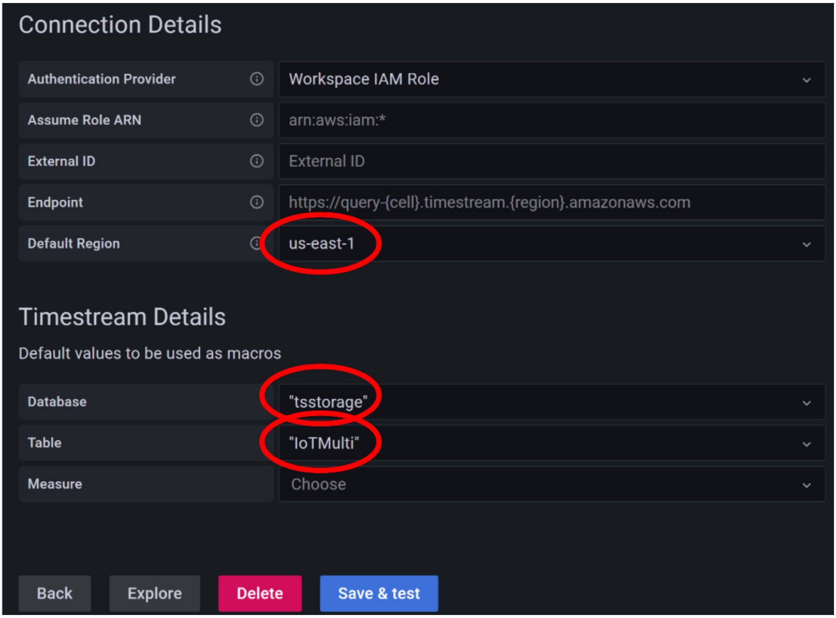

aws
Configuration
Organization: Main Org.
Data sources
Search by name or type
Add data source
Amazon Timestream
Amazon Timestream | default
aws
Configuration
Data sources
Users
Teams

Figure 10.41 – Add a Timestream data source

1. Choose the Timestream data source, configure your region, and select a database and table. Finally, click on Test and Save & test :



Connection Details

Authentication Provider: Workspace IAM Role

Assume Role ARN: arn:aws:iam:*

External ID: External ID

Endpoint: https://query-{cell}.timestream.{region}.amazonaws.com

Default Region: us-east-1

Timestream Details

Default values to be used as macros

Database: "tsstorage"

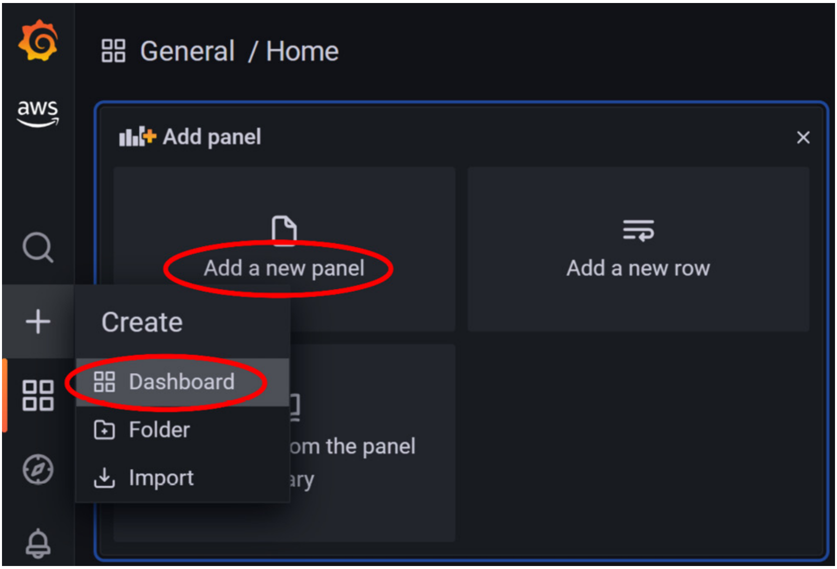
Table: "IoTMulti"

Measure: Choose

Back
Explore
Delete
Save & test

Figure 10.42 – Configure a data source

1. To configure a dashboard, from the left menu, create a dashboard and then a new panel, as shown in the following figure:



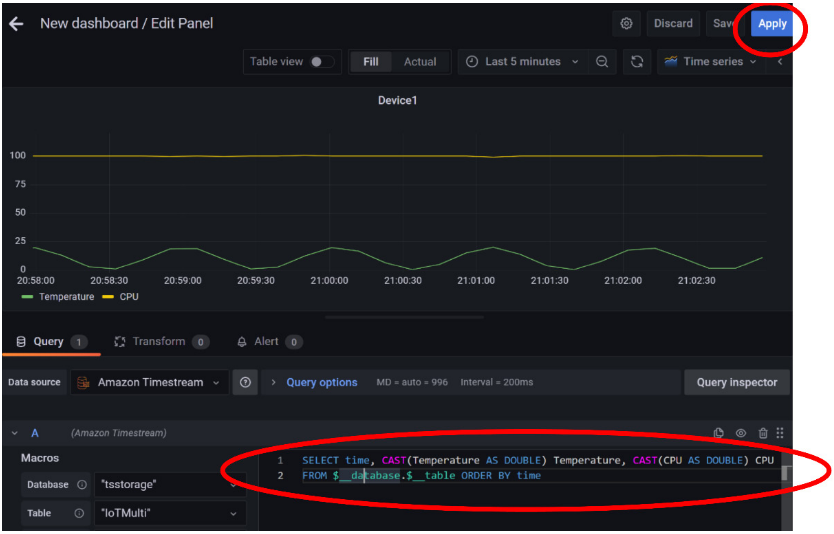
aws
General / Home
Add panel
Add a new panel
Add a new row
Create
Dashboard
Folder
Import
om the panel
ary

Figure 10.43 – Build a Grafana dashboard

In the query section of the panel, configure the source of our measures:

```
SELECT  time, CAST(Temperature AS
DOUBLE) Temperature, CAST(CPU AS
DOUBLE) CPU FROM $__database.$__table
ORDER BY time DESC LIMIT 300
```

1. Choose Timeseries visualization type and click on Apply :


New dashboard / Edit Panel
Discard
Save
Apply
Table view
Fill
Actual
Last 5 minutes
Time series
Device1
100
75
50
25
0
20:58:00
20:58:30
20:59:00
20:59:30
21:00:00
21:00:30
21:01:00
21:01:30
21:02:00
21:02:30
Temperature
CPU
Query 1
Transform 0
Alert 0
Query options
Query inspector
A
(Amazon Timestream)
Macros

Data source: Amazon Timestream

MD = auto = 996   Interval = 200ms

Database: "tsstorage"

Table: "IoTMulti"

```
SELECT time, CAST(Temperature AS DOUBLE) Temperature, CAST(CPU AS DOUBLE) CPU
FROM $__database.$__table ORDER BY time
```

Figure 10.44 – Build our first dashboard

If everything has been done correctly, you will see the data displayed on the screen.

You can also change the type of view. In the following case, we changed it to Gauge . If the edge is sending data, you will see the data changing every minute:

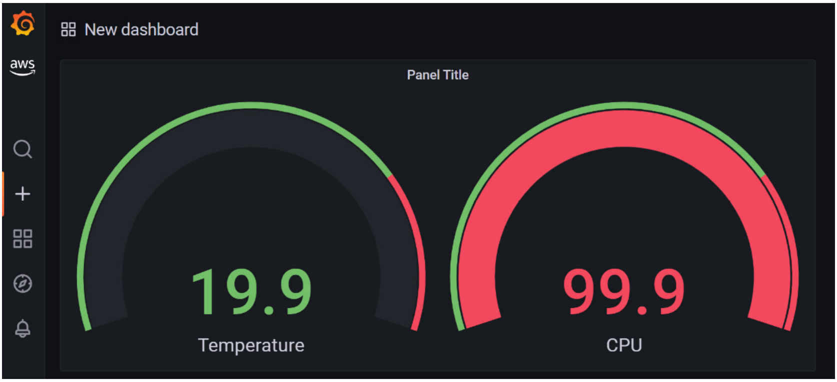


Figure 10.45 – Real-time data

Data visualization and dashboard development is the last step of our journey with AWS services. Now, we have an architecture ready to manage the entire flow from the edge to the operational phase.

About visualization

Grafana is an excellent tool and is easy to configure; however, if you must manage many assets and many sensors, you will have to adopt a much more tailored strategy for your needs. Do not forget that adopting a good asset governance and naming convention strategy will save your business.

Summary

In this chapter, we finalized an end-to-end system from data to visualization; we saved the data and visualized the historical series with Grafana. In the next chapter, we will discover the Azure IoT offering.

11

Performing a Practical Industrial IoT Solution with Azure

In the previous chapters (*Chapter 9* and *Chapter 10*), we looked at how to build an IIoT platform with AWS and a custom solution. We explored some of the most important concepts, including storage, time series, microservices, protocols, and data analytics. The purpose of this chapter and the next is to build an IIoT solution with Azure.

In this chapter, we will discover the most popular Azure cloud components that can be applied to IIoT. We will cover the following:

- Azure IoT
- Connecting a device with IoT Edge
- Connecting a device with the Azure Edge SDK

- Azure Service Bus

In the next chapter (*Chapter 12*), we will continue to explore the Azure ecosystem implementing the operational data flow leveraging the following technologies:

- Azure Event Hubs
- Azure Cosmos DB
- Azure Stream Analytics
- Azure Synapse
- Building visualizations with Data Explorer and Grafana

Toward the end of this chapter, we will learn about the differences between the solutions proposed.

Technical requirements

To follow along with this chapter, we need to download and install the following prerequisites:

- Docker : https://www.docker.com/
- Git : https://git-scm.com/downloads
- Python 3.9+ : https://www.python.org/downloads/

Azure IoT ecosystem

Azure IoT is a platform designed by Microsoft to connect multiple devices, enable telemetry, store measures, run and develop analytics, and visualize results. The following are some key components of Azure IoT:

- IoT Hub
- Stream Analytics
- Azure IoT Edge and SDK

Other services that we can use to build our IIoT architecture (not specific to IIoT) are the following:

- Machine Learning (ML) analytics
- Grafana
- Azure Cosmos DB
- Azure Synapse

The following diagram shows the architecture proposed by us to build IIoT infrastructure on Azure:

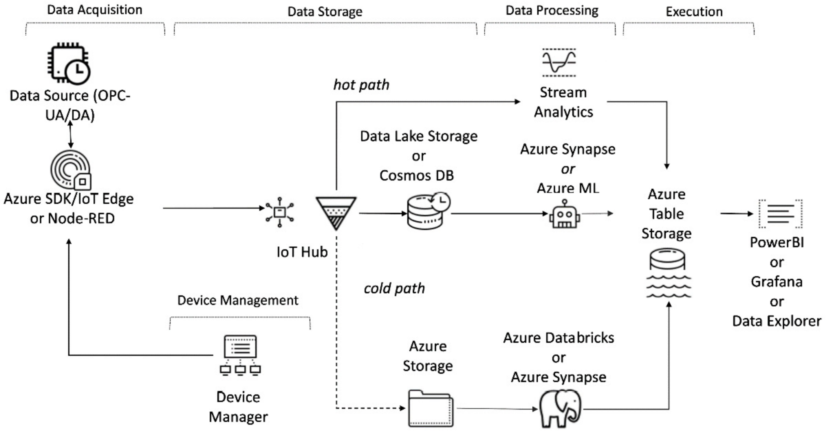

Data Acquisition
Data Storage
Data Processing
Execution
Data Source (OPC-UA/DA)
Azure SDK/IoT Edge or Node-RED
IoT Hub
hot path
Stream Analytics
Data Lake Storage or Cosmos DB
Azure Synapse or Azure ML
Azure Table Storage
PowerBI or Grafana or Data Explorer
cold path
Device Management
Device Manager
Azure Storage
Azure Databricks or Azure Synapse

Figure 11.1 – The proposed Azure IIoT architecture

Data is acquired through the Azure IoT Edge or Azure IoT SDK and sent to Azure IoT Hub. Data can be processed with low latency Stream Analytics, stored in a time series database called Cosmos DB, or stored in Azure Storage services. It can then be processed by Azure ML analytics or Synapse. Finally, we can use Grafana, Data Explorer, or Power BI for fast visualization of the data.

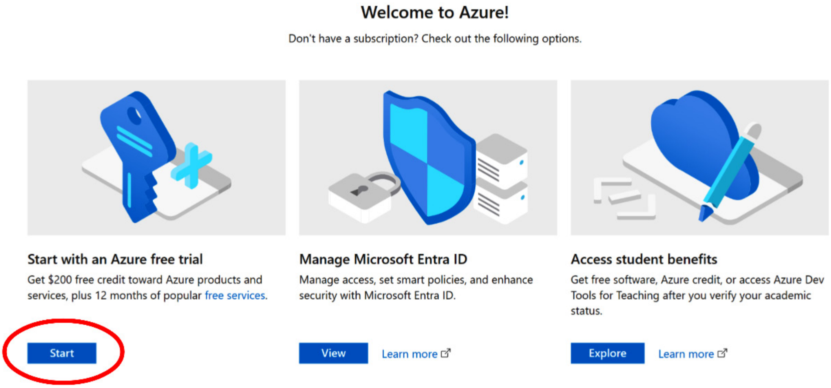
The code for this chapter is available in the official repository at https://github.com/PacktPublishing/Hands-On-Industrial-Internet-of-Things-Second-

Edition .

In this section, we explored how to implement an IIoT architecture with Azure. In the next section, we will see how to implement it in practice.

Registering for Azure IoT

To start working with Azure IoT, we need to register for a free Azure account. As the first step, go to https://azure.microsoft.com/en-us/free/ and click on the Start button shown in the following figure:



Welcome to Azure!

Don't have a subscription? Check out the following options.

Start with an Azure free trial

Get $200 free credit toward Azure products and services, plus 12 months of popular free services.

Manage Microsoft Entra ID

Manage access, set smart policies, and enhance security with Microsoft Entra ID.

Access student benefits

Get free software, Azure credit, or access Azure Dev Tools for Teaching after you verify your academic status.

Figure 11.2 – Starting an Azure free trial

To be able to get a free Azure account, you need to have a Microsoft account. If needed, you can create one. Once the account is set up, Azure will ask you for credit card details and two-factor authentication.

When the procedure has been completed, you can access the Azure dashboard by opening the browser to https://portal.azure.com/ .

The next step is to enable the IoT Hub console.

Azure IoT Hub

IoT Hub is the middleware that is used to register and manage devices. IoT Hub supports the MQTTS, HTTP, HTTPS, WebSocket, and AMQP protocols.

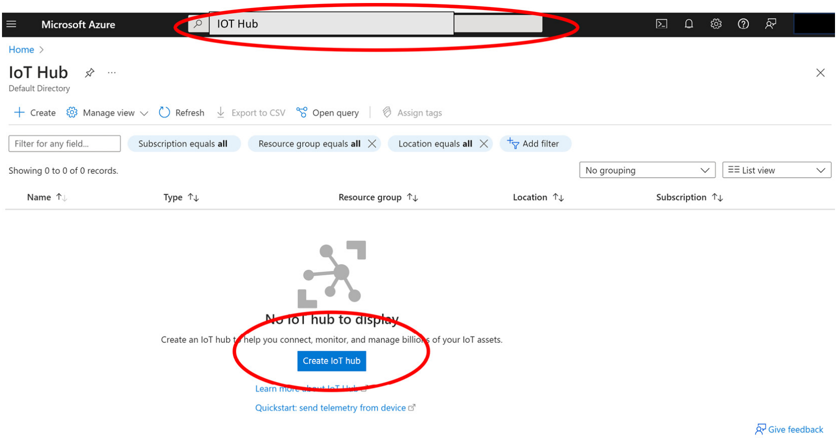
From a security point of view, Azure IOT supports the following methods:

- SAS token : This is based on private key sharing.
- X.509 certificates : As we saw in *Chapter 9* , the device is configured with an X.509 certificate.
- Device provisioning service (DPS): DPS automates the provisioning of IoT devices. It is very useful if you need to configure thousands of devices.

- Symmetric key authentication or shared access key authentication : Central IOT Hub and edges share the same secret key.
- Azure Active Directory authentication : This allows IoT devices to authenticate using Active Directory.

In our exercise, we will use the X.509 authentication mechanism.

To start working with IoT Hub, we need to search for `IoT Hub` using the search bar (see *Figure 11 .3*). Once you have found it, click on the Create IoT hub button. For convenience, we can pin IoT Hub to the dashboard by clicking on the Pin button:


Microsoft Azure
IOT Hub
Home >
IoT Hub
Default Directory
Create
Manage view
Refresh
Export to CSV
Open query
Assign tags
Filter for any field...
Subscription equals all
Resource group equals all
Location equals all
Add filter
No grouping
List view
Name
Type
Resource group
Location
Subscription
Create IoT hub
Learn more about IoT Hub
Quickstart: send telemetry from device
Give feedback

Showing 0 to 0 of 0 records.

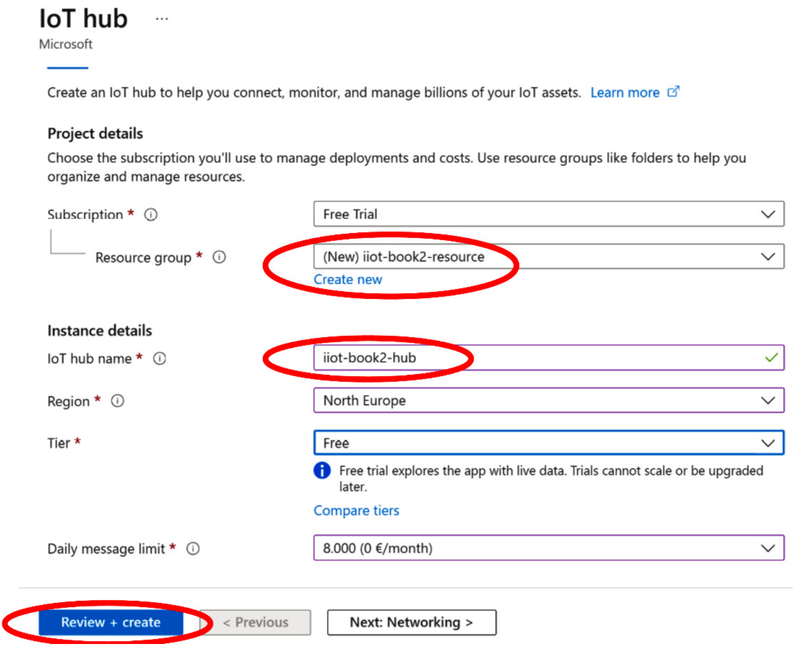
No IoT hub to display

Create an IoT hub to help you connect, monitor, and manage billions of your IoT assets.

Figure 11.3 – Accessing the Azure IoT Hub console

We can then start creating our hub by following these steps:

1. After clicking on the Create IoT hub button, provide the name of the resource (`iiot- book2-resources`) and the name of the hub (`iiot-book2-hub`).
2. Choose the Free tier and the suitable region, and click on Review + create , as shown in the following screenshot:



IoT hub

Microsoft

Create an IoT hub to help you connect, monitor, and manage billions of your IoT assets. Learn more

Project details

Choose the subscription you'll use to manage deployments and costs. Use resource groups like folders to help you organize and manage resources.

Subscription * : Free Trial

Resource group * : (New) iiot-book2-resource

Instance details

IoT hub name * : iiot-book2-hub

Region * : North Europe

Tier * : Free

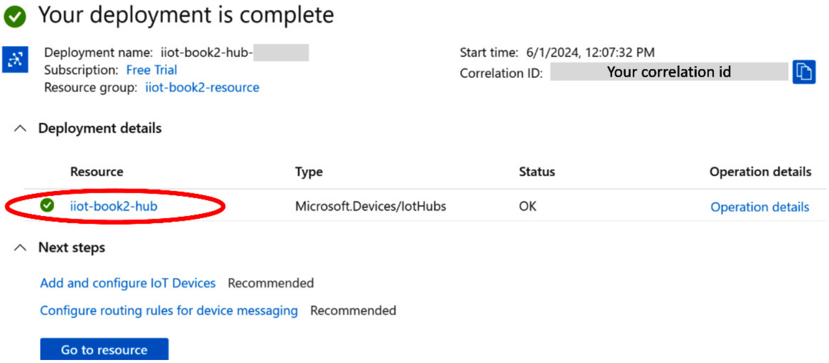
Free trial explores the app with live data. Trials cannot scale or be upgraded later.

Daily message limit * : 8.000 (0 €/month)

Figure 11.4 – Creating our IoT hub

1. Confirm the setup on the next screen and, finally, click on the Create button.

Azure will create an instance of the IoT Hub in a few minutes. If everything goes well, we will be able to go to the dashboard, click on the IoT Hub instance that we just created, and see the Overview page and the hostname of our IoT Hub:



Your deployment is complete

Deployment name: iiot-book2-hub-[redacted]
Subscription: Free Trial
Resource group: iiot-book2-resource

Start time: 6/1/2024, 12:07:32 PM
Correlation ID: Your correlation id

Deployment details

Resource	Type	Status	Operation details
iiot-book2-hub	Microsoft.Devices/IotHubs	OK	Operation details

Next steps

Add and configure IoT Devices Recommended
Configure routing rules for device messaging Recommended

Figure 11.5 – Deployment status of IOT Hub

In our case, the hostname will be **`iiot-book2-hub.azure-devices.net`**. Take note of this name; we will reuse it in the next sessions during edge configuration.

Event Hubs

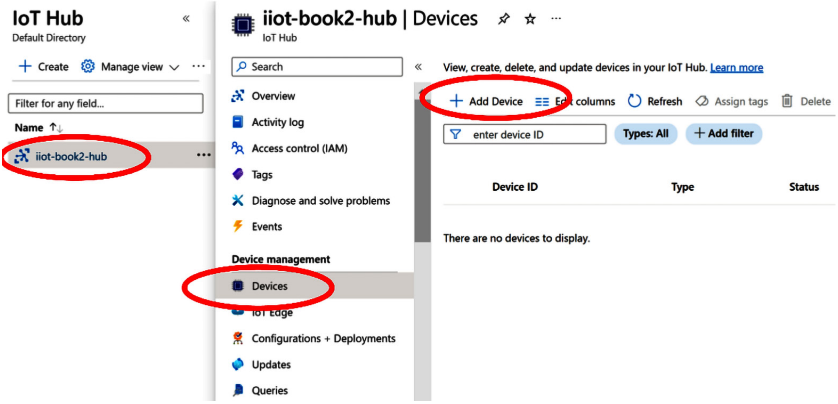
IoT Hub is very efficient for telemetry, but if you need massive data flow, look

> *at* ***Event Hubs*** *(https://learn.microsoft.com/en-us/azure/event-hubs/event-hubs-about). Event Hubs can be an alternative to IoT Hub if you want to inject massive amounts of data such as images, videos, or big files.*

Registering a new device

We can now register our first device. From the dashboard, perform the following steps:

1. Click on the IoT Hub instance labeled iiot-book2-hub .
2. Click the Devices tab.
3. Click on the Add Device button and provide a name for our device – for example, `device1` :


IoT Hub
Default Directory
Create
Manage view
Filter for any field...
Name
iiot-book2-hub
iiot-book2-hub | Devices
IoT Hub
Search
Overview
Activity log
Access control (IAM)
Tags
Diagnose and solve problems
Events
Device management
Devices
IoT Edge
Configurations + Deployments
Updates
Queries
Add Device
Edit columns
Refresh
Assign tags
Delete
enter device ID
Types: All
Add filter
Device ID
Type
Status

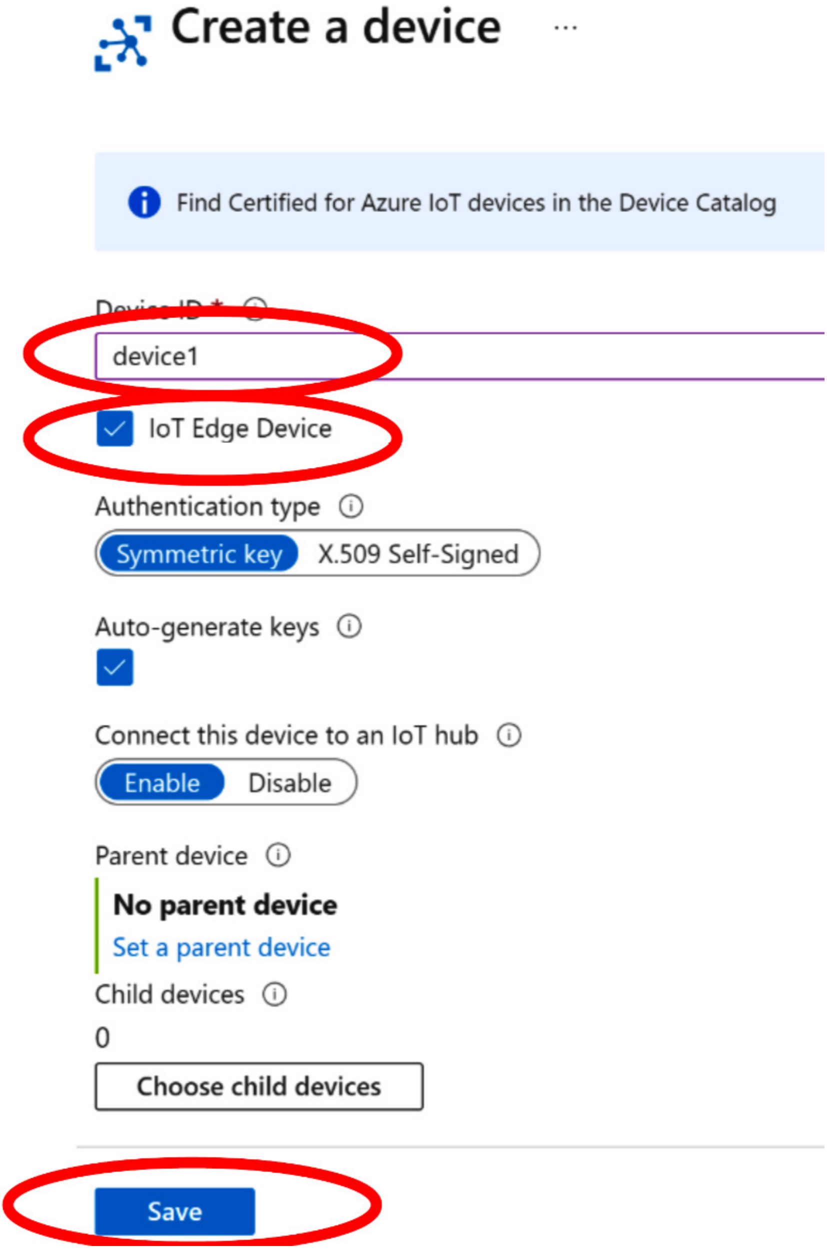
View, create, delete, and update devices in your IoT Hub. Learn more

There are no devices to display.

Figure 11.6 – Creating an IOT device

1. Check the IOT Edge Device checkbox. We will use a SAS authentication token to establish a secure channel between IoT Edge and IoT Hub.

2. Finally, click on the Save button:



Create a device

Find Certified for Azure IoT devices in the Device Catalog

Device ID *: device1

☑ IoT Edge Device

Authentication type: Symmetric key (selected) | X.509 Self-Signed

Auto-generate keys: ☑

Connect this device to an IoT hub: Enable (selected) | Disable

Parent device: No parent device

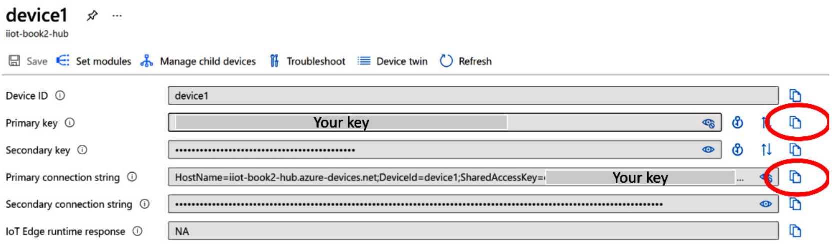
Child devices: 0

Set a parent device
Choose child devices
Save

Figure 11.7 – Steps to configure a new IoT device

in Azure IoT Hub

After creating the device, we can get the host connection string from the Device Details page. Copy the SAS key and the connection string, as shown in the following figure:


device1
iiot-book2-hub
Save
Set modules
Manage child devices
Troubleshoot
Device twin
Refresh

Device ID: device1
Primary key: Your key
Secondary key: ••
Primary connection string: HostName=iiot-book2-hub.azure-devices.net;DeviceId=device1;SharedAccessKey= Your key ...
Secondary connection string: ••
IoT Edge runtime response: NA

Figure 11.8 – The device details page

We are ready to start ingesting data from the sensors.

Azure IoT Edge

> *Note that we need to enable the IoT Edge flag because we will reuse this device during the Azure IoT Edge exercise. This flag enables the remote management of Azure IOT Edge components.*

Building a custom edge

In the previous section, we configured a new device on IOT Hub. Now, we are going to implement our edge. We will reuse, as usual, Node-RED and the Azure IOT Edge Python SDK.

Building the edge with Node-RED

As we have seen in the previous chapters, we will reuse the Node-RED device to start sending data from the OPC-UA server simulator (a simulated boiler with temperature and CPU signals) to IoT Hub. Let's reuse the previous code; the only upgrade is to install Azure IoT Hub support.

In the command line, run the following command:

```
$ npm install node-red-contrib-azure-
iot-hub
```

Alternatively, you can reuse the Dockerfile created in the previous chapters (*Chapter 4* , *Chapter 8* , and *Chapter 9*) and add the following line to the Dockerfile:

```
$ npm install node-red-contrib-azure-
iot-hub
```

The code for this exercise is available in the official repository at https://github.com/PacktPublishing/Hands-On-Industrial-Internet-of-Things-Second-Edition .

Start Node-RED and the OPC-UA server as seen in the previous chapters. If you are using Docker Compose, build the images with the following command:

```
$ docker compose build
```

With the following command, start the containers:

```
$ docker compose up
```

Finally, connect to http://localhost:1880/ .

Now, we need to build the following data flow:

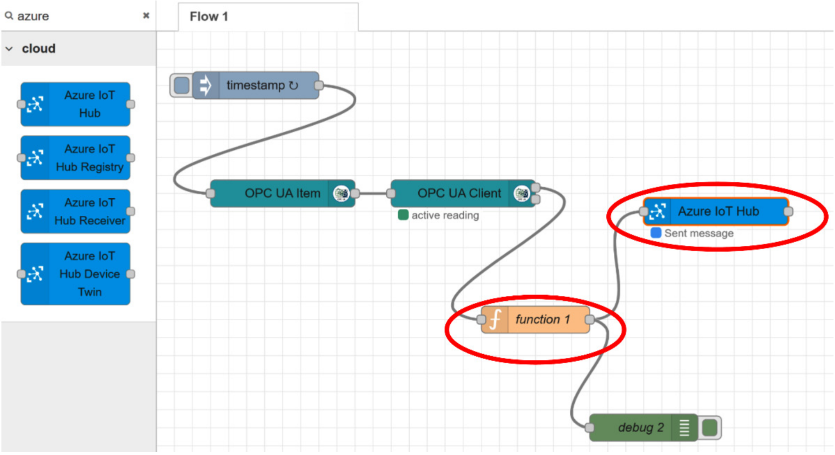

azure
cloud
Flow 1
Azure IoT Hub
Azure IoT Hub Registry
Azure IoT Hub Receiver
Azure IoT Hub Device Twin
timestamp ↻
OPC UA Item
OPC UA Client
active reading
Azure IoT Hub
Sent message
function 1
debug 2

Figure 11.9 – Node-RED to Azure IoT Hub data flow

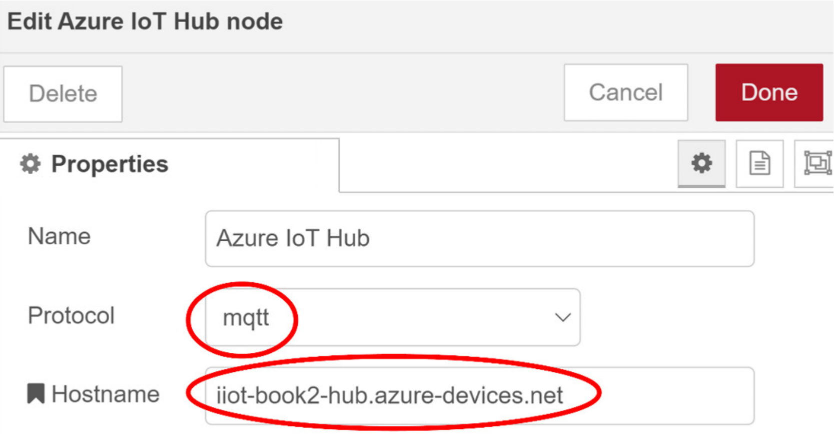
Compared to the data flow we saw previously, we have added two new components. Let's learn about them in detail.

1. *Component 1* – Add a function with the following code:

```
function generateGuid() {
   return
Math.random().toString(36).substring(2,
15) +
Math.random().toString(36).substring(2,
15);
}
msg.payload = {
  "message-id": generateGuid(),
  "deviceId": "device1",
  "key": "YOUR KEY",
    "protocol": "mqtt",
    "data": {"temperature":
  msg.payload}
  }
  return msg;
```

Replace **`YOUR KEY`** with the SAS key generated earlier in the *Registering a new device* section. Azure IoT Hub wants a payload in a specific format: the first function generates the UID of the message. Then, we need to authenticate the device with the **`deviceId`** and **`key`** properties. Finally, we can define the data payload by setting the **`data`** property.

2. *Component 2* – Add the Azure IOT Hub component and configure the host to **`iiot-book2-hub.azure-devices.net`** (or the name of your IoT Hub). Switch the protocol to MQTT , as shown in the following figure:


Edit Azure IoT Hub node
Delete
Cancel
Done
Properties

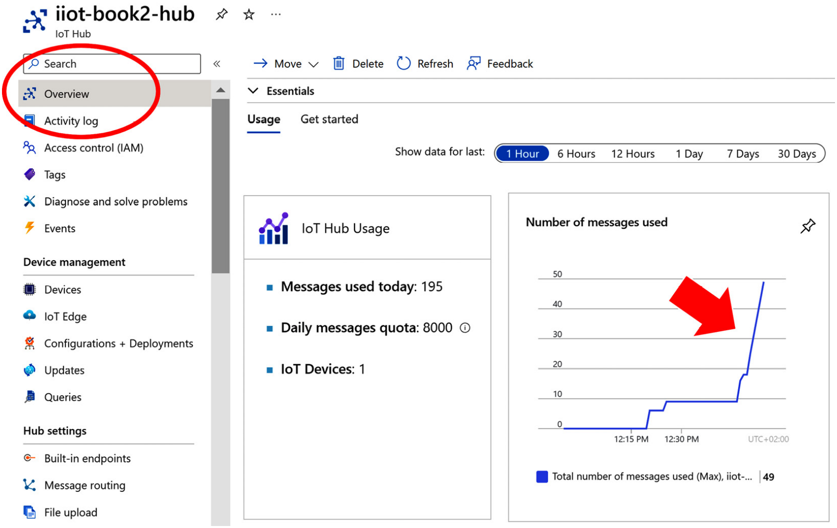
Name: Azure IoT Hub

Protocol: mqtt

Hostname: iiot-book2-hub.azure-devices.net

Figure 11.10 – Configuring the Azure IoT Hub component on Node-RED

Finally, click on the IoT Hub created before, and click on Overview to see the data flowing into Azure IoT Hub:


iiot-book2-hub
IoT Hub
Search
Overview
Activity log
Access control (IAM)
Tags
Diagnose and solve problems
Events
Device management
Devices
IoT Edge
Configurations + Deployments
Updates
Queries
Hub settings
Built-in endpoints
Message routing
File upload
Move
Delete
Refresh
Feedback
Essentials
Usage
Get started
Show data for last:
1 Hour
6 Hours
12 Hours
1 Day
7 Days
30 Days
IoT Hub Usage
Number of messages used
50
40
30
20
10
0
12:15 PM
12:30 PM
UTC+02:00
Total number of messages used (Max), iiot-... 49

- **Messages used today**: 195
- **Daily messages quota**: 8000
- **IoT Devices**: 1

Figure 11.11 – Data flowing into Azure IoT Hub

Node-RED allows us to build a device with very few lines of code, but if you want more robust control, we can develop our device using Python. In the next section, we will build the same device based on Python.

Building the device with Python

Azure supports the IoT SDK to develop proprietary devices. It is possible to use .NET, Java, Python, and ANSI C. In this exercise, we will develop our simple edge in Python.

The code for this exercise is available in the official repository at https://github.com/PacktPublishing/Hands-On-Industrial-Internet-of-Things-Second-Edition .

Building the Python code

Create a new folder directory called `python-device` . In this directory, create a Python file called

`OPCUABridge.py` .

Import the required packages:

```
import os,random,time, sys, datetime,
opcua, uuid, asyncio
from azure.iot.device import Message
from azure.iot.device.aio import
IoTHubModuleClient
```

Now, we can write the function to get data from the OPC-UA server and send it to IoT Hub. The code is quite similar to what we have done in *Chapter 9* :

```
async def send_telemetry(clientOPCUA,
clientIOT,
msg_data_format='{{"temperature":
{temperature},"cpu": {cpu}}}'):
    myvarC =
clientOPCUA.get_node("ns=1;s=CPU").get_valu
    myvarT =
clientOPCUA.get_node("ns=1;s=Temperature").
    msg_data_formatted =
msg_data_format.format(temperature=myvarT,
cpu=myvarC)
    message = Message(data =
msg_data_formatted,
                      content_encoding='utf
                      content_type='applica
json',
                        message_id=str(uuid.u
```

Note the content type and encoding to notify IoT Hub of the encoding of the body. Do not forget it or IoT Hub will encode the body in Base64 format. Indeed, you can use this mechanism to send unstructured data such as files or images.

Moreover, you can add an extra parameter capable of sending an alarm if the CPU is over 99%. Here is how:

```
    if myvarC > 99:
        message.custom_properties["CPUAlert
= "true"
    else:
        message.custom_properties["CPUAlert
= "false"
```

Now, we can send the data:

```
    clientIOT.send_message(message)
```

Define the main function and start the connection. Replace the connection string with your connection string:

```
async def main():
    CONNECTION_STRING = "YOUR
CONNECTION STRING"
    clientIOT =
IoTHubModuleClient.create_from_connection_s
    await clientIOT.connect()
```

Initialize the OPC UA connection:

```
        clientOPCUA =
opcua.Client("opc.tcp://opc-ua-
server:26543")
        clientOPCUA.connect()
```

Start to send data:

```
        while True:
            await
asyncio.gather(send_telemetry(clientOPCUA,
clientIOT))
            time.sleep(10)
```

Finally, define the **`main`** method using **`asyncio`** support:

```
if __name__ == '__main__':
    asyncio.run(main())
```

Building and starting the containers

The next step is to build Docker images. Create a new file called **`docker-compose-python.yml`** and add the following two services:

```
version: '2'
services:
  python-device:
    image: iiot-book-2/python-device
    build: ./python-device
    tty: True
    environment:
        -
IOTHUB_DEVICE_CONNECTION_STRING="YOUR
CONNECTION STRING"
  opc-ua-server:
    build: ./node-opcua-sampleserver
```

To build the image of the Python edge, create a new Dockerfile in the **`python-device`** folder just created and copy and paste the following line of code:

```
FROM python:3.9.19
RUN python3 -m pip install opcua
crypto cryptography azure-iot-device
COPY ./OPCUABridge.py /
CMD ["python3", "OPCUABridge.py"]
```

Now, we can build the images with `docker compose build`:

```
$ docker compose -f .\docker-compose-
python.yml build
```

When all the images are ready, start the containers; from the command line, run the following:

```
$ docker compose -f .\docker-compose-
python.yml up
```

Starting the edge from a local PC (optional)

Instead of using Docker, you can run the edge from your PC (not suggested).

Install the required packages and, from the command line, run the following:

```
$ pip install opcua crypto cryptography
azure-iot-device
```

Start the OPCUA server simulator. When the simulator is ready, we can start our edge. From the command line, run the following:

```
$ python OPCUABridge.py
```

The client will send some data from the OPC UA server. We also added a property to indicate when the CPU is greater than 99%. Implementing our own custom edge through the Azure IoT SDK offers maximum flexibility. In some circumstances, however, we need a more modular edge, with strong support for remote management and deployment. Azure offers Azure IoT Edge. Next, we will learn how to implement an IoT edge using the official Azure IoT Edge.

Azure IoT Edge

Azure IoT Edge (`aziot`) is the Edge gateway and Edge computer of Azure. In the previous section, we implemented a simple Edge gateway with the Azure Device SDK and Python. Azure IoT Edge simplifies the development process and enhances the computation capabilities of the Edge, distributing business logic to devices through standard containers and monitoring them from the cloud.

Some features of IoT Edge are as follows:

- Computation : With Azure IoT Edge, we can deploy Azure Stream Analytics jobs and Azure ML analytics on the Edge to perform simple or advanced actions. The Azure IoT Edge runtime supports C#, Node.js, Java, Python, and C.
- Gateway : Azure IoT Edge can connect sensors through MQTT, AMQP, and HTTP. However, it needs additional modules to support OPC UA, OPC DA, and Modbus.

Azure IoT Edge also supports a store and forward feature in its offline mode. Azure IoT Edge allows us to save data locally for a long period of time, in case we have an intermittent connection.

The operating systems supported by Azure IoT Edge are Linux and Windows. Azure IoT Edge leverages Docker strongly. It is based on three components:

- Azure IoT Edge runtime : This orchestrates the containers.
- IoT Edge hub : This is responsible for instantiating modules.
- IoT Edge agent : This is a proxy for IoT Hub. It exposes the same protocol endpoints as IoT Hub.

The core concept of Azure IoT Edge is to utilize Docker components for custom modules as well: while the Azure IoT Edge runtime takes care of *orchestrating* the modules, the Azure IoT Edge hub is responsible for *managing* the custom modules. Modules can be developed using the Azure IoT Edge dev tool; the entire continuous integration/ continuous deployment (CI/CD) process is controlled on the cloud side. Developers can build a Docker module and push it to the Docker registry, then they need to define this registry on IoT Hub and deploy the module (see the next figure). However, there are many other modules developed by third parties that can be deployed directly from the cloud; in the *Azure IoT Edge applied to the industrial sector* section, we will see how to deploy a module for OPC-UA.

The following figure shows the basic idea under the CI/CD Azure IoT Edge pipeline:

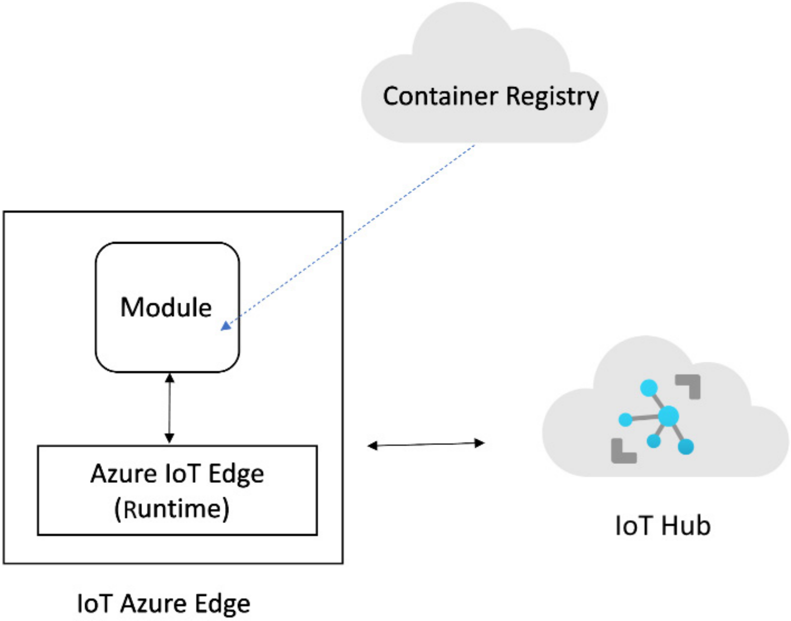

Container Registry
Module
Azure IoT Edge
(Runtime)
IoT Hub
IoT Azure Edge

Figure 11.12 – Azure IoT Edge CI/CD

In the next section, we will see how to use Azure IoT Edge to send data to the cloud. As shown in the following figure, for our exercise, we will leverage Docker also for the IoT Edge runtime:

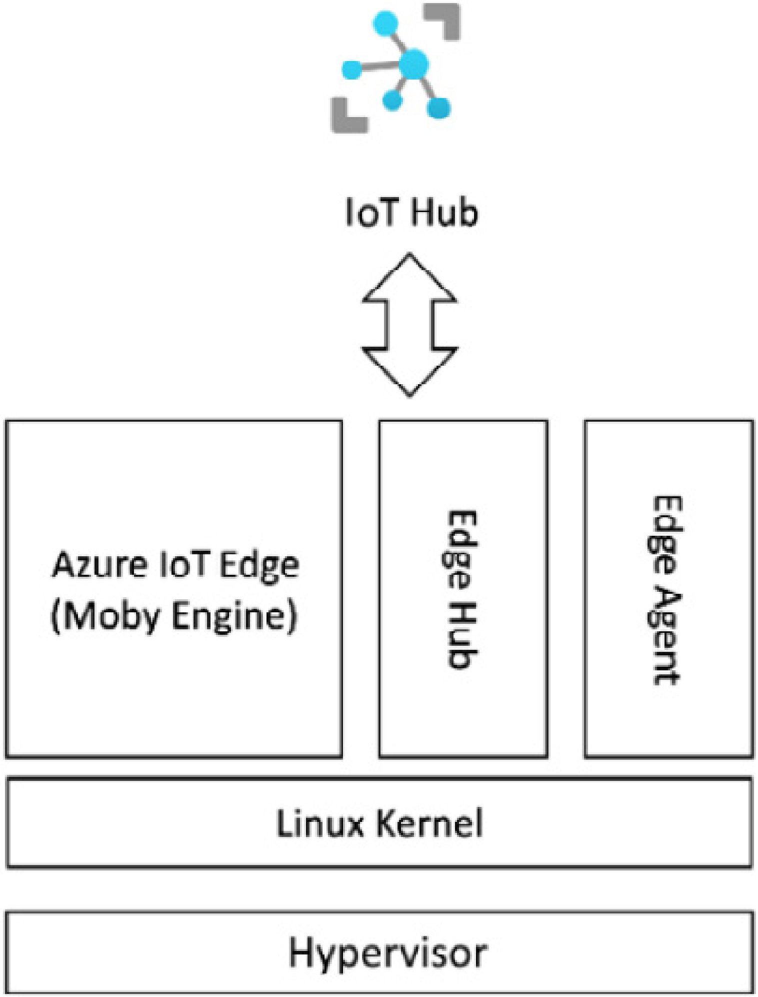

IoT Hub
Azure IoT Edge
(Moby Engine)
Edge Hub
Edge Agent
Linux Kernel
Hypervisor

Figure 11.13 – Azure IoT Edge architecture

Starting Azure IoT Edge

To start IoT Edge, you can use Linux or Windows; in our exercise, we are going to use Ubuntu 22.04.

Windows version

> *If you are working on Windows, you can follow the instructions listed at https://learn.microsoft.com/en-us/azure/iot-edge/?view=iotedge-1.5 . Alternatively, just start **Windows Subsystem for Linux** (**WSL**): from PowerShell, run the* `ubuntu` *command.*

Now, we need to install some prerequisites. From Bash, run the following lines:

```
$ sudo apt-get -y update
$ sudo apt-get -y install wget nano
```

Add the Microsoft official Ubuntu 22.04 repository. From Bash, run the following lines:

```
$ sudo wget https://
packages.microsoft.com/config/
ubuntu/22.04/packages-microsoft-prod.deb
-O packages-microsoft-prod.deb
$ sudo dpkg -i packages-microsoft-
prod.deb
```

Then, install the Moby engine and Azure IoT Edge. From Bash, run the following lines:

```
$ sudo apt-get -y update
$ sudo apt-get -y upgrade
$ sudo apt-get -y install moby-engine
moby-cli aziot-edge
```

Configure and restart Docker. From Bash, run the following lines:

```
$ sudo echo {"log-driver": "local"} >> /
etc/docker/daemon.json
$ sudo systemctl restart docker
```

Now, we can configure the connection to IoT Hub by adding the connection string. From Bash, execute the following lines:

```
$ iotedge config mp --connection-string
<your connection string>
$ sudo sudo iotedge config apply
```

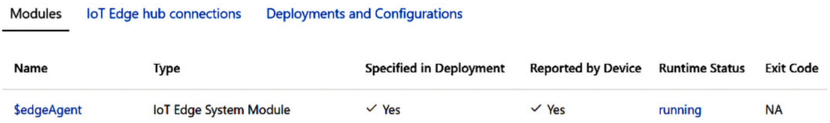
Here, `connection string` is the string from IoT Hub.

Finally, check the agent status:

```
$ sudo iotedge list
```

You should at least see the agent status as running .

Lastly, you can check in the IoT Hub from Devices | device1 | Modules , as shown here:

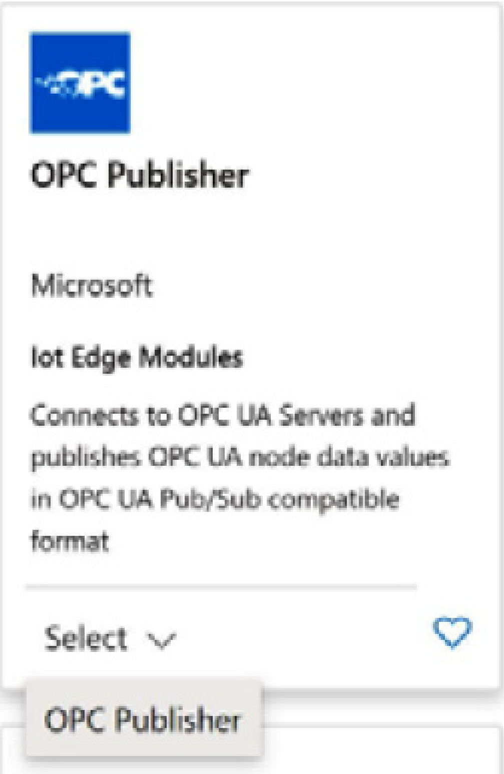


Name	Type	Specified in Deployment	Reported by Device	Runtime Status	Exit Code
$edgeAgent	IoT Edge System Module	✓ Yes	✓ Yes	running	NA

Figure 11.14 – Checking the IoT Edge agent working on Azure IoT Hub

Azure IoT Edge applied to the industrial sector

As mentioned earlier, to work in the industrial sector, we need to deploy additional modules to support standard data acquisition, such as OPC UA. Azure doesn't provide an official provider for OPC-UA, but we can leverage Azure Marketplace. Go to IoT Hub | Devices | device1 | Set modules , and add a new module from Marketplace, as follows:



OPC Publisher

Microsoft

Iot Edge Modules

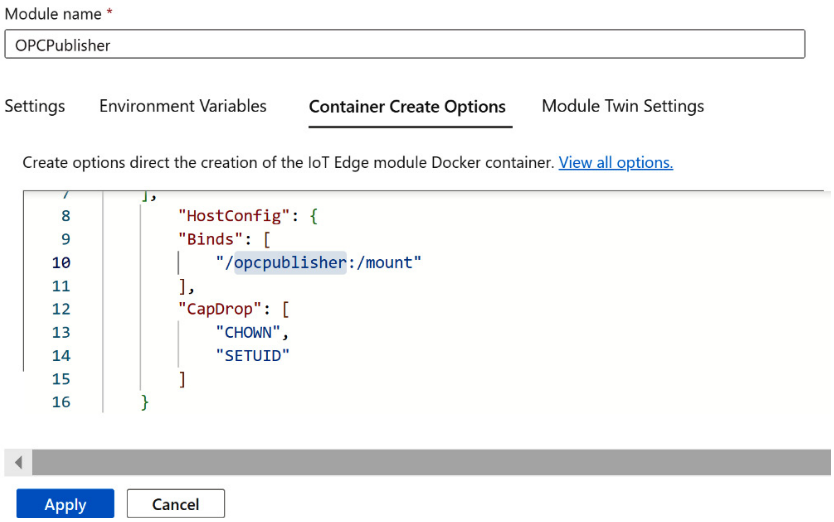
Connects to OPC UA Servers and publishes OPC UA node data values in OPC UA Pub/Sub compatible format

Figure 11.15 – OPC Publisher from Azure Marketplace

Then, configure the remote host to **`localhost`**. Create a file on the host machine, **`/opcpublisher/publishednodes.json`**, and add the following lines to the JSON configuration file:

```
[
    {
        "EndpointUrl": "opc.tcp://
localhost:26543",
        "OpcNodes": [
            {
                "DisplayName":
"temperature",
                "Id": "ns=1;s=CPU",
                "ConditionHandling": {
                    "UpdateInterval":
10
                }
            }
        ]
    }
]
```

In Container Create Options , bind the directory of the localhost to Docker, as shown ahead, then click on the Apply button:



Module name *

OPCPublisher

Settings
Environment Variables
Container Create Options
Module Twin Settings

Create options direct the creation of the IoT Edge module Docker container. View all options.

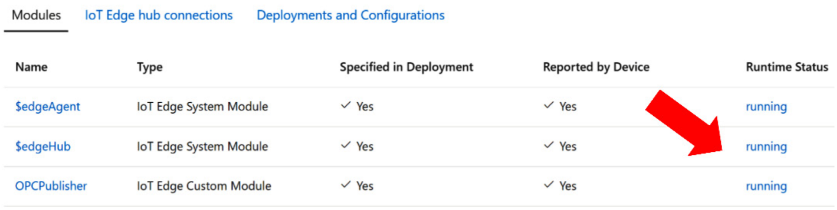
```
    ],
    "HostConfig": {
    "Binds": [
        "/opcpublisher:/mount"
    ],
    "CapDrop": [
        "CHOWN",
        "SETUID"
    ]
}
```

Apply
Cancel

Figure 11.16 – Binding the folder

On the next screen, click on the Create and deploy button. You can go to https://azure.github.io/Industrial-IoT/opc-publisher/ for a complete list of parameters for this module.

If everything went well, you will see the data flowing to the cloud and the modules correctly running, as shown in the following figure:



Name	Type	Specified in Deployment	Reported by Device	Runtime Status
$edgeAgent	IoT Edge System Module	✓ Yes	✓ Yes	running
$edgeHub	IoT Edge System Module	✓ Yes	✓ Yes	running
OPCPublisher	IoT Edge Custom Module	✓ Yes	✓ Yes	running

Figure 11.17 – Checking the status of modules deployed on an Azure IoT device

You can also double-check the status by executing the following command on the edge:

```
$ sudo iotedge list
```

The expected output should be similar to the following:

```
NAME             STATUS           DESCRIPTIO
OPCPublisher     running          Up 9
minutes     mcr.microsoft.com/iotedge/
opc-publisher:latest
edgeAgent        running          Up 9
minutes     mcr.microsoft.com/
azureiotedge-agent:1.5
edgeHub          running          Up 9
minutes     mcr.microsoft.com/
azureiotedge-hub:1.5
```

As you can see, Azure IoT Edge is truly flexible as it is based on Docker modules, which are easily deployable.

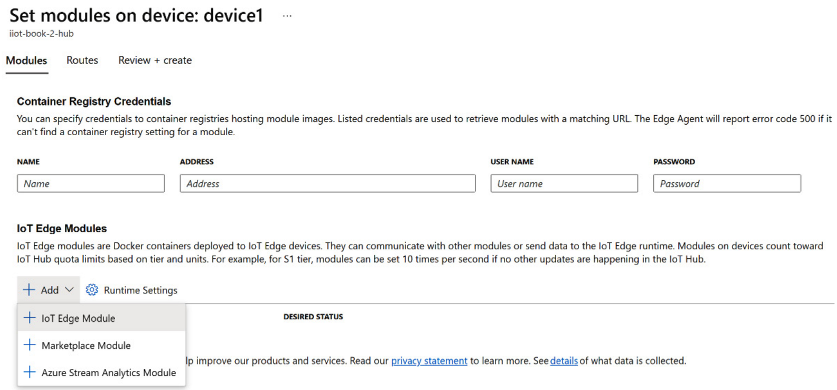
Building a custom module for Azure IoT Edge

Azure allows you to develop a custom module leveraging the Azure IoT Edge dev tool (see https://github.com/Azure/iotedgedev). If you want to develop a module, install `iotedgedev` or use the Docker container provided by Microsoft.

When you are ready, you can leverage the CI/CD pipeline offered by Microsoft and configure it from

IoT Hub | Devices | device1 | Set modules .

Then, configure the Docker registry where you have pushed the image, as follows:


Set modules on device: device1 ···
iiot-book-2-hub
Modules
Routes
Review + create
NAME
ADDRESS
USER NAME
PASSWORD
Name
Address
User name
Password
Add
Runtime Settings
IoT Edge Module
Marketplace Module
Azure Stream Analytics Module
DESIRED STATUS

Container Registry Credentials

You can specify credentials to container registries hosting module images. Listed credentials are used to retrieve modules with a matching URL. The Edge Agent will report error code 500 if it can't find a container registry setting for a module.

IoT Edge Modules

IoT Edge modules are Docker containers deployed to IoT Edge devices. They can communicate with other modules or send data to the IoT Edge runtime. Modules on devices count toward IoT Hub quota limits based on tier and units. For example, for S1 tier, modules can be set 10 times per second if no other updates are happening in the IoT Hub.

lp improve our products and services. Read our privacy statement to learn more. See details of what data is collected.

Figure 11.18 – Building a custom module and deploying it to IoT Edge

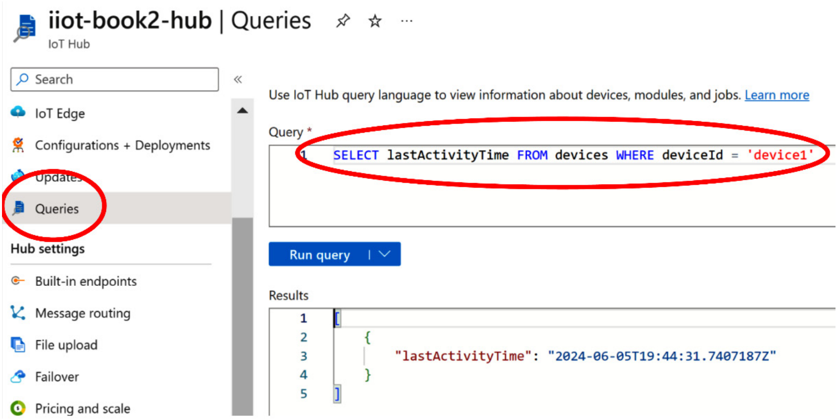
Using this method, you can also deploy Azure Stream Analytics (ASA) on the edge. We will work with ASA in *Chapter 12* .

Device twins

Azure IoT keeps the status of the edge in an internal database. We can ask the status of the device using the query tool of IoT Hub. From the IoT Hub just created, click on Queries and run the following query:

```
SELECT lastActivityTime FROM devices
WHERE deviceId = 'device1'
```

IoT Hub will retrieve the last sent message and it will show you the last update. See the following figure for more details:


iiot-book2-hub | Queries
IoT Hub
Search
IoT Edge
Configurations + Deployments
Updates
Queries
Hub settings
Built-in endpoints
Message routing
File upload
Failover
Pricing and scale
Query *
Run query
Results

Use IoT Hub query language to view information about devices, modules, and jobs. Learn more

```
SELECT lastActivityTime FROM devices WHERE deviceId = 'device1'
```

```
[
    {
        "lastActivityTime": "2024-06-05T19:44:31.7407187Z"
    }
]
```

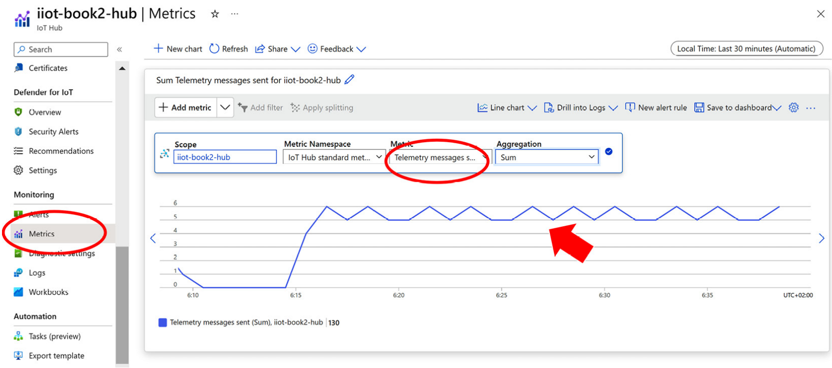
Figure 11.19 – Query language of device twins

You can visit https://learn.microsoft.com/en-us/azure/iot-hub/query-twins for more information about the query language.

Monitoring messages sent through

IoT Hub Metrics

We can also monitor the messages that are sent from the Metrics tab of IoT Hub, as shown in the following screenshot:


iiot-book2-hub | Metrics
IoT Hub
Search
Certificates
Defender for IoT
Overview
Security Alerts
Recommendations
Settings
Monitoring
Alerts
Metrics
Diagnostic settings
Logs
Workbooks
Automation
Tasks (preview)
Export template
New chart
Refresh
Share
Feedback
Local Time: Last 30 minutes (Automatic)
Add metric
Add filter
Apply splitting
Line chart
Drill into Logs
New alert rule
Save to dashboard
6
5
4
3
2
1
0
6:10
6:15
6:20
6:25
6:30
6:35
UTC+02:00

Sum Telemetry messages sent for iiot-book2-hub

Scope: iiot-book2-hub
Metric Namespace: IoT Hub standard met...
Metric: Telemetry messages s...
Aggregation: Sum

Telemetry messages sent (Sum), iiot-book2-hub | 130

Figure 11.20 – The Metrics page in IoT Hub

To monitor the messages, click on Metrics and select Telemetry messages sent .

In this section, we understood how to set up the edge device to read data from the OPC-UA data source and send data to IoT Hub. In the next section, we will learn how to manage data and configure the data flow.

Showing messages sent through Service Bus

To visualize the raw data of our message, we can connect IoT Hub to Service Bus. This exercise is not mandatory to finalize our architecture, but sometimes it might be useful to debug our messages – for example, in all those cases in which the edge does not send the data in the expected format or sends it empty or corrupted.

Service Bus is a native Azure service based on the pub-sub mechanism. So, in the next step, we will define a queue where we send our data and we will connect to IoT Hub, but the first step is to locate Service Bus. In the Azure portal search bar, search for **`service bus`** , as shown in the following figure, and click on Service Bus :

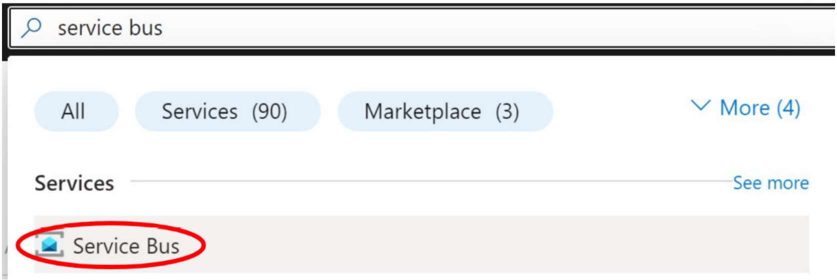
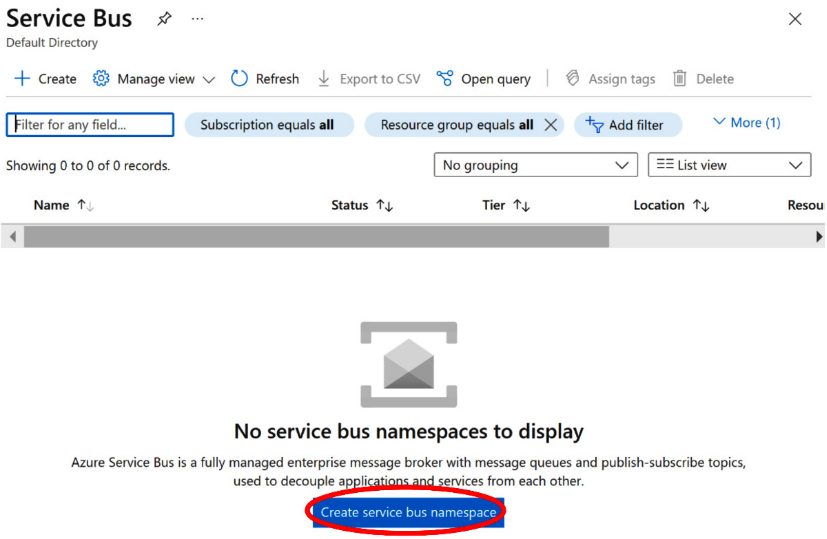

service bus
All
Services (90)
Marketplace (3)
More (4)
Services
See more
Service Bus

Figure 11.21 – Locating Service Bus in the Azure portal

Then, click on Create service bus namespace :


Service Bus
Default Directory
Create
Manage view
Refresh
Export to CSV
Open query
Assign tags
Delete
Filter for any field...
Subscription equals **all**
Resource group equals **all**
Add filter
More (1)
No grouping
List view
Name
Status
Tier
Location
Resou
Create service bus namespace

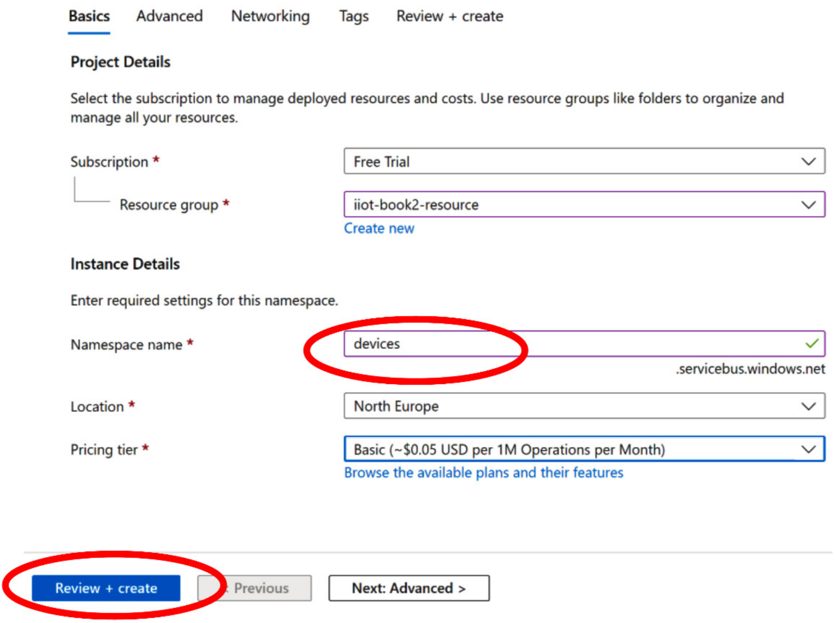
Showing 0 to 0 of 0 records.

No service bus namespaces to display

Azure Service Bus is a fully managed enterprise message broker with message queues and publish-subscribe topics, used to decouple applications and services from each other.

Figure 11.22 – Creating a Service Bus namespace

Next, define the name, for example, `devices` , as shown in the following figure:


Basics
Advanced
Networking
Tags
Review + create
Review + create
< Previous
Next: Advanced >

Project Details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * Free Trial

Resource group * iiot-book2-resource

Create new

Instance Details

Enter required settings for this namespace.

Namespace name * devices ✓ .servicebus.windows.net

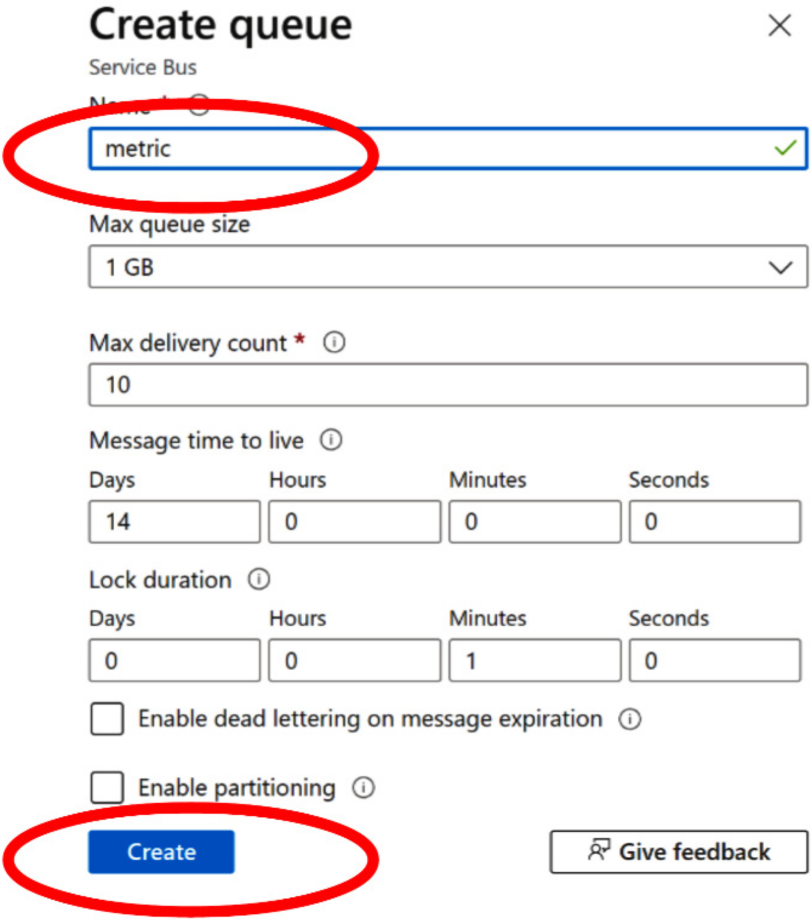
Location * North Europe

Pricing tier * Basic (~$0.05 USD per 1M Operations per Month)

Browse the available plans and their features

Figure 11.23 – Defining the name of the namespace

Finally, create a queue (call it `metric`), as follows. We want to keep our data for no longer than 14 days:



Create queue

Service Bus

Name * ⓘ

metric

Max queue size

1 GB

Max delivery count * ⓘ

10

Message time to live ⓘ

Days	Hours	Minutes	Seconds
14	0	0	0

Lock duration ⓘ

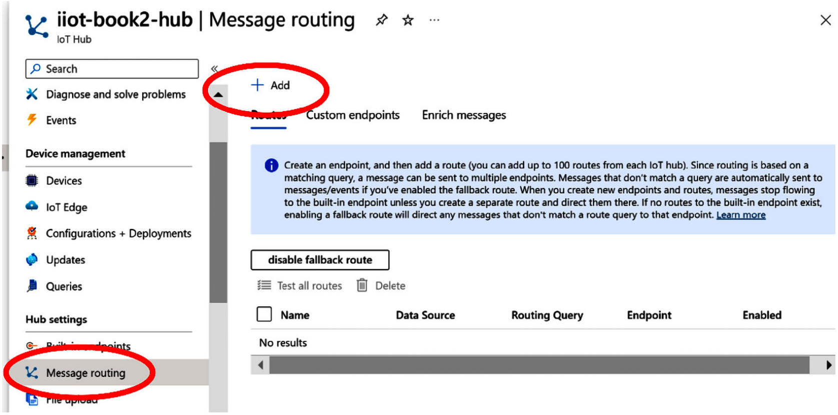
Days	Hours	Minutes	Seconds
0	0	1	0

☐ Enable dead lettering on message expiration ⓘ

☐ Enable partitioning ⓘ

Figure 11.24 – Creating a queue for our data from IoT Hub

Now, we can ask IoT Hub to send data to Service Bus. From the left-side menu of IoT Hub, you have to add message routing, as follows:


iiot-book2-hub | Message routing
IoT Hub
Search
Diagnose and solve problems
Events
Device management
Devices
IoT Edge
Configurations + Deployments
Updates
Queries
Hub settings
Built-in endpoints
Message routing
File upload
+ Add
Routes
Custom endpoints
Enrich messages
disable fallback route
Test all routes
Delete

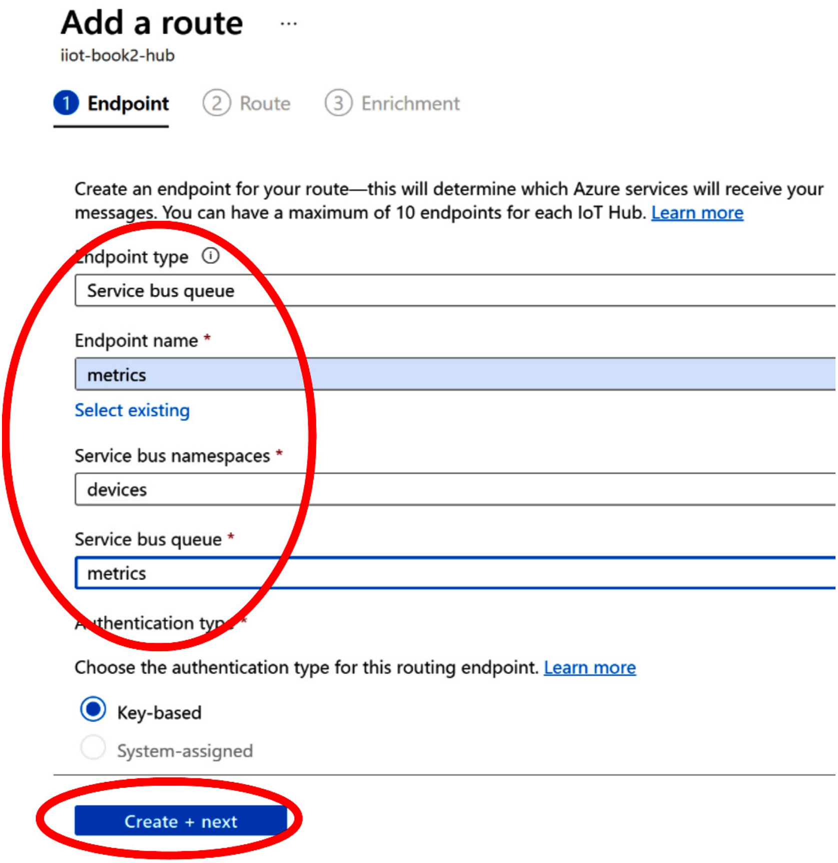
Create an endpoint, and then add a route (you can add up to 100 routes from each IoT hub). Since routing is based on a matching query, a message can be sent to multiple endpoints. Messages that don't match a query are automatically sent to messages/events if you've enabled the fallback route. When you create new endpoints and routes, messages stop flowing to the built-in endpoint unless you create a separate route and direct them there. If no routes to the built-in endpoint exist, enabling a fallback route will direct any messages that don't match a route query to that endpoint. Learn more

☐ Name	Data Source	Routing Query	Endpoint	Enabled
No results				

Figure 11.25 – Adding new message routing

Set a route for the Service Bus queue created before.

The following figure shows details of this step:


Add a route ···
iiot-book2-hub
1 Endpoint
2 Route
3 Enrichment

Create an endpoint for your route—this will determine which Azure services will receive your messages. You can have a maximum of 10 endpoints for each IoT Hub. Learn more

Endpoint type: Service bus queue

Endpoint name *: metrics

Select existing

Service bus namespaces *: devices

Service bus queue *: metrics

Authentication type *

Choose the authentication type for this routing endpoint. Learn more

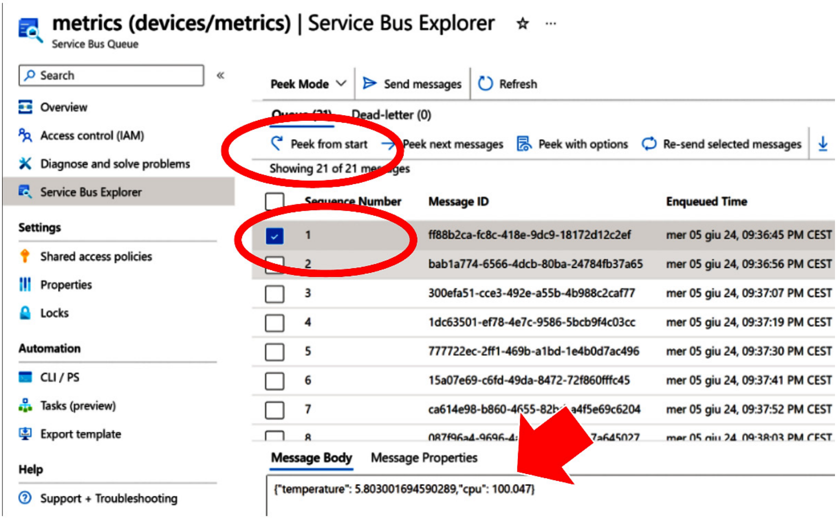
● Key-based

○ System-assigned

Create + next

Figure 11.26 – Defining message routing

Do remember to start the edge (our simulated boiler) to send data to the cloud. On Service Bus | Service Bus Explorer , click on Peek from start to see the temperature set by our (simulated) boiler, as follows:



metrics (devices/metrics) | Service Bus Explorer
Service Bus Queue
Search
Overview
Access control (IAM)
Diagnose and solve problems
Service Bus Explorer
Settings
Shared access policies
Properties
Locks
Automation
CLI / PS
Tasks (preview)
Export template
Help
Support + Troubleshooting
Peek Mode
Send messages
Refresh
Queue (21)
Dead-letter (0)
Peek from start
Peek next messages
Peek with options
Re-send selected messages
Showing 21 of 21 messages
Message Body
Message Properties

☐	Sequence Number	Message ID	Enqueued Time
☑	1	ff88b2ca-fc8c-418e-9dc9-18172d12c2ef	mer 05 giu 24, 09:36:45 PM CEST
☐	2	bab1a774-6566-4dcb-80ba-24784fb37a65	mer 05 giu 24, 09:36:56 PM CEST
☐	3	300efa51-cce3-492e-a55b-4b988c2caf77	mer 05 giu 24, 09:37:07 PM CEST
☐	4	1dc63501-ef78-4e7c-9586-5bcb9f4c03cc	mer 05 giu 24, 09:37:19 PM CEST
☐	5	777722ec-2ff1-469b-a1bd-1e4b0d7ac496	mer 05 giu 24, 09:37:30 PM CEST
☐	6	15a07e69-c6fd-49da-8472-72f860fffc45	mer 05 giu 24, 09:37:41 PM CEST
☐	7	ca614e98-b860-4655-82b[illegible]a4f5e69c6204	mer 05 giu 24, 09:37:52 PM CEST
☐	8	087f96a4-9696-4[illegible]7a645027	mer 05 giu 24, 09:38:03 PM CEST

```
{"temperature": 5.803001694590289,"cpu": 100.047}
```

Figure 11.27 – Using Service Bus to see data sent from the edge

We have concluded our service of connecting an edge device and sending data to the cloud. In *Chapter 12* , we will work with our data directly on the cloud.

Summary

In this chapter, we have explored the most common functionalities of Azure IoT. We have developed three types of edge – an edge based on the Azure

SDK, an external edge, and Azure IoT Edge – and looked at their advantages and disadvantages. In *Chapter 12* , we will finalize our exercise and implement the operational data flow from IoT Hub to the final user.

12

Implementing an Industrial IoT Data Flow with Azure

In the previous chapters, we looked at how to build an IIoT platform with Azure IoT Hub and Azure IoT Edge. The purpose of this chapter is to discover the other components that Azure offers to manage the data flow into the cloud.

We will cover the following:

- Azure Cosmos DB
- Azure Stream Analytics
- Azure Synapse
- Building visualizations with Data Explorer and Grafana

Toward the end of this chapter, we will learn about the differences between the solutions proposed.

Technical requirements

To follow along with this chapter, we need to download and install the following prerequisites:

- Docker : https://www.docker.com/
- Git : https://git-scm.com/downloads
- Python 3.9+ : https://www.python.org/downloads/

Architecture of the proposed exercise

In *Chapter 11* , we implemented the edge-to-cloud data flow using the Azure IoT Edge SDK and Azure IoT Hub. Let's recall, in the next figure, the architecture proposed to build IIoT infrastructure on Azure :

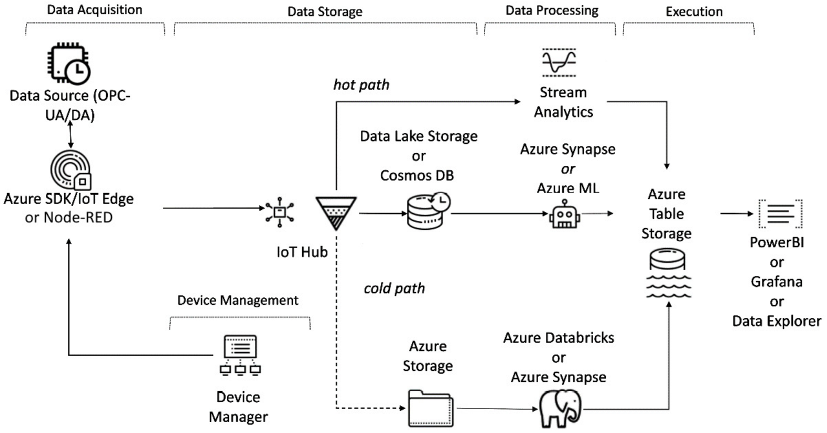

Data Acquisition
Data Storage
Data Processing
Execution
Data Source (OPC-UA/DA)
Azure SDK/IoT Edge or Node-RED
IoT Hub
hot path
Stream Analytics
Data Lake Storage or Cosmos DB
Azure Synapse or Azure ML
Azure Table Storage
PowerBI or Grafana or Data Explorer
cold path
Device Management
Device Manager
Azure Storage
Azure Databricks or Azure Synapse

Figure 12.1 – The proposed Azure IIoT architecture

When measures are sent to Azure IoT Hub, data can be processed with low latency Stream Analytics, stored in a time series database called Azure Cosmos DB, or stored in Azure Storage services. The data can subsequently be analyzed using Azure ML analytics or Synapse. For rapid visualization, tools such as Grafana, Data Explorer, or Power BI can be employed.

The code for this chapter is available in the official repository at https://github.com/PacktPublishing/Hands-On-Industrial-Internet-of-Things-Second-

Edition .

In this section, we explored how to implement an IIoT architecture with Azure. In the next section, we will see how to implement it in practice.

Setting the data flow

When data is sent to IoT Hub, we need to handle and manage it. Common use cases for data management are the following:

- Data storage : We can save our telemetry data and/or device status in various databases: Azure Data Lake or Cosmos DB. This data can be used for visualization, processing, and execution later.
- Data processing :
    - Real-time data analytics : We can apply real-time data processing and aggregation
    - Rule-based message routing : We can define actions to take when data is received in the cloud
- Data transformation : Data can be enriched and/or transformed:
    - Integration with Azure applications : We can integrate data with custom applications via custom endpoints

    - Event notifications and processing : We can redirect messages to Azure Event Grid or Service Bus

- Visualization : We can show data and events.

We will describe these use cases in the next sections.

Data storage

In this section, we will see how to store our data; however, we need to accomplish some prerequisites.

Azure offers a long list of services for both SQL-based, NoSQL, and third-party storage; for IoT purposes, we can leverage the following:

- The Blob Storage service accessible via the Storage Browser
- Azure Time Series Insights (deprecated in 2024)
- Cosmos DB

Storage browser and Blob Storage

Generally, you can access Table and Blob Storage through a storage account. The service allows you to define custom tables, containers, file shares, and queues. We will learn about Table storage in the *Azure Stream Analytics* section.

For now, we just need to configure the storage account to use it in the next sections.

Before creating a storage service, Azure requires you to create a storage account. To create storage accounts, locate the Storage accounts service from the search bar and create a new one. You can see the following figure for reference:

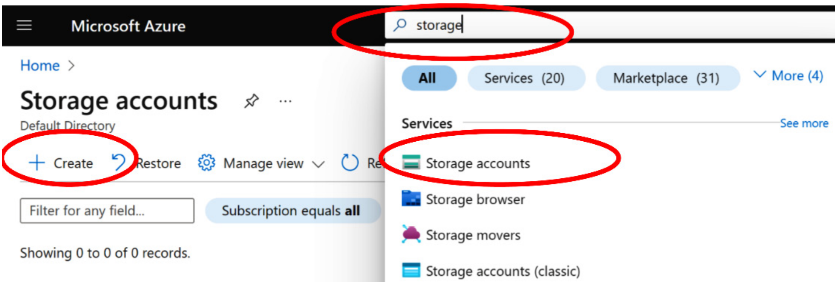
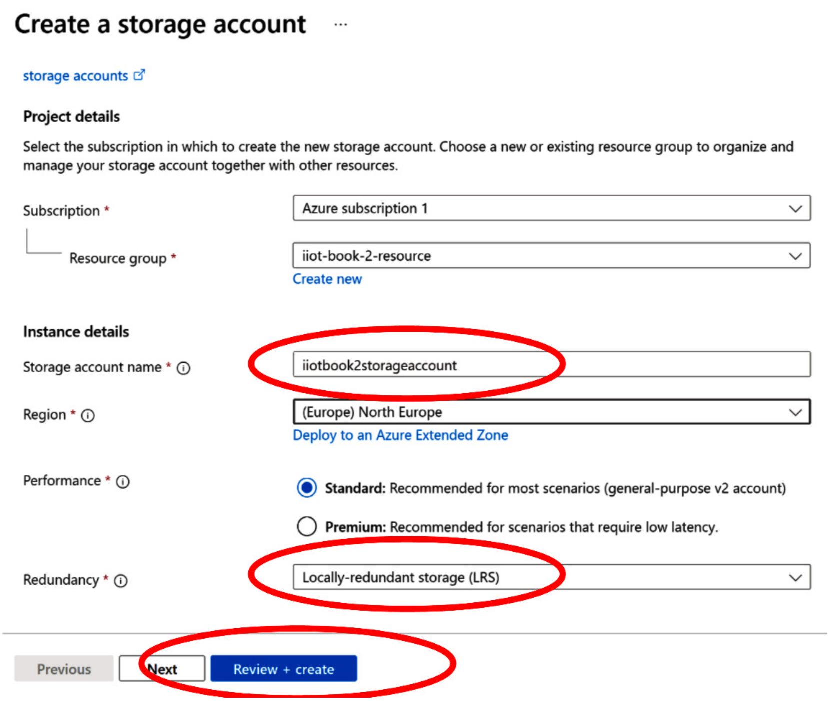

Microsoft Azure
storage
Home >
Storage accounts
Default Directory
+ Create
Restore
Manage view
All
Services (20)
Marketplace (31)
More (4)
Services
See more
Storage accounts
Storage browser
Storage movers
Storage accounts (classic)
Filter for any field...
Subscription equals all
Showing 0 to 0 of 0 records.

Figure 12.2 – Creating a storage account

Then, provide a name. For our exercise, Locally-redundant storage (LRS) is enough. Finally, click on Review + create , as follows:



Create a storage account ...

storage accounts

Project details

Select the subscription in which to create the new storage account. Choose a new or existing resource group to organize and manage your storage account together with other resources.

Subscription *: Azure subscription 1

Resource group *: iiot-book-2-resource

Create new

Instance details

Storage account name *: iiotbook2storageaccount

Region *: (Europe) North Europe

Deploy to an Azure Extended Zone

Performance *:

● **Standard:** Recommended for most scenarios (general-purpose v2 account)

○ **Premium:** Recommended for scenarios that require low latency.

Redundancy *: Locally-redundant storage (LRS)

Figure 12.3 – Creating a locally-redundant storage account

Now, we can define our tables and queues; we will reuse the storage account in the *Azure Stream Analytics* section when working with analytics. In the next section, we will explore Cosmos DB.

Cosmos DB

Azure Cosmos DB can store edge messages as JSON documents in the hot data store. To work with

Cosmos DB, search for Cosmos DB in the Azure portal and create a new instance of it, as shown in the following figure:

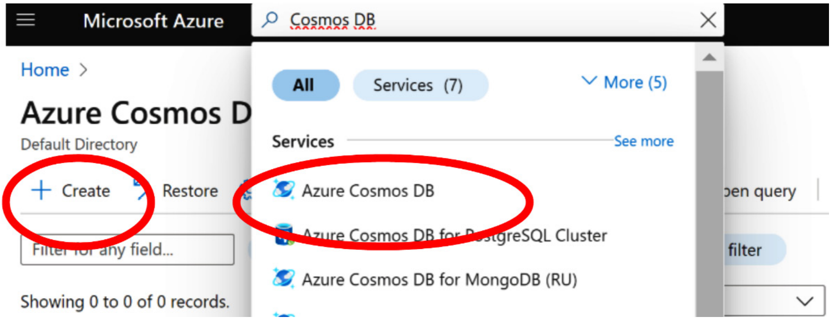
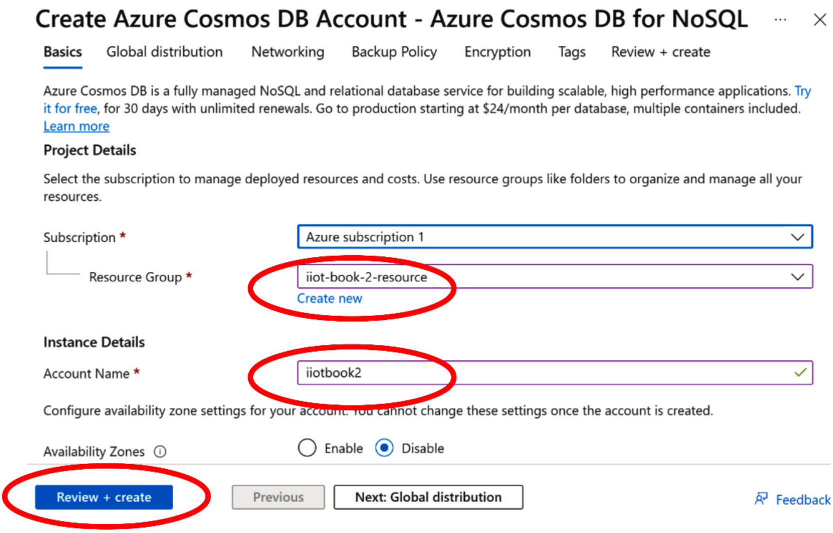

Microsoft Azure
Cosmos DB
Home >
Azure Cosmos D
Default Directory
+ Create
Restore
Filter for any field...
Showing 0 to 0 of 0 records.
All
Services (7)
More (5)
Services
See more
Azure Cosmos DB
Azure Cosmos DB for PostgreSQL Cluster
Azure Cosmos DB for MongoDB (RU)
pen query
filter

Figure 12.4 – Creating a Cosmos DB account

Cosmos DB supports several use cases. In our case, we'll choose Azure Cosmos DB for NoSQL . Then, set the resource group and provide your account name, as shown in the following figure:


Create Azure Cosmos DB Account - Azure Cosmos DB for NoSQL
Basics
Global distribution
Networking
Backup Policy
Encryption
Tags
Review + create
Review + create
Previous
Next: Global distribution
Feedback

Azure Cosmos DB is a fully managed NoSQL and relational database service for building scalable, high performance applications. Try it for free, for 30 days with unlimited renewals. Go to production starting at $24/month per database, multiple containers included. Learn more

Project Details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription *: Azure subscription 1

Resource Group *: iiot-book-2-resource (Create new)

Instance Details

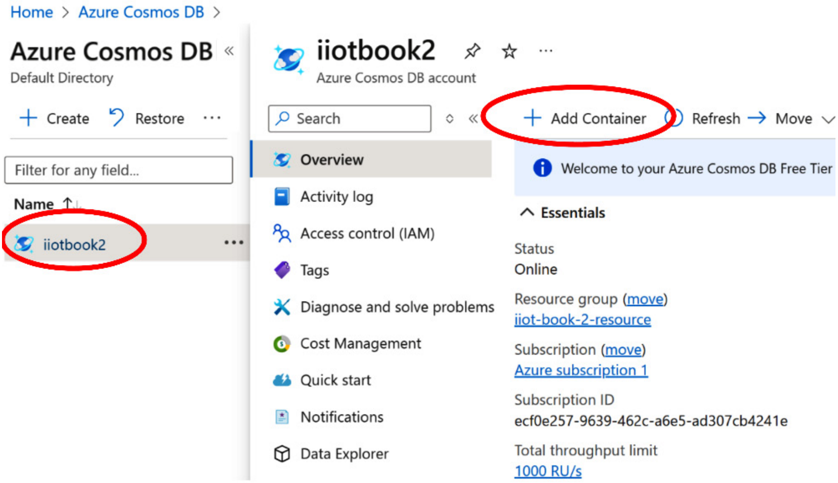
Account Name *: iiotbook2

Configure availability zone settings for your account. You cannot change these settings once the account is created.

Availability Zones: ○ Enable ● Disable

Figure 12.5 – Creating a Cosmos DB account

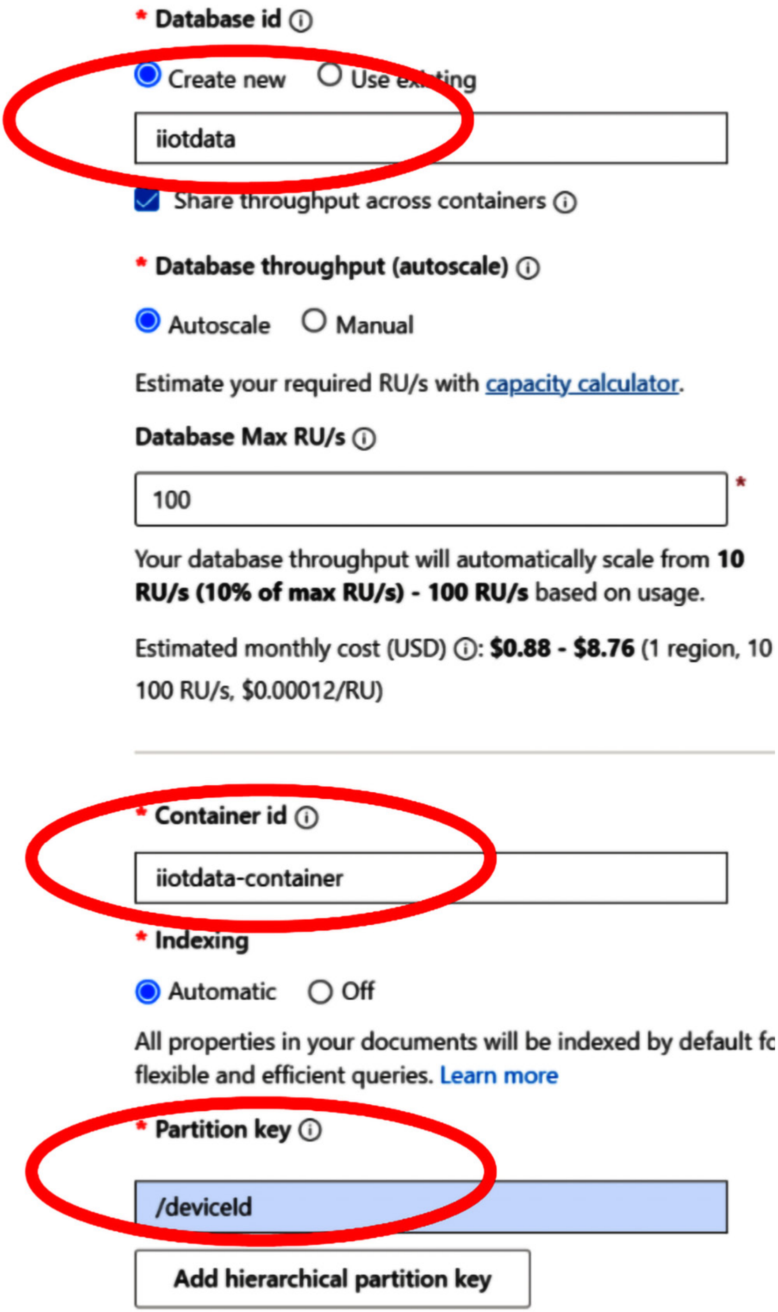
In the final step, we must create the container instance. From the Cosmos DB account created before, add a container, as shown in the following figure:


Home > Azure Cosmos DB >
Azure Cosmos DB
Default Directory
Create
Restore
Filter for any field...
Name
iiotbook2
iiotbook2
Azure Cosmos DB account
Search
Add Container
Refresh
Move
Overview
Activity log
Access control (IAM)
Tags
Diagnose and solve problems
Cost Management
Quick start
Notifications
Data Explorer
Welcome to your Azure Cosmos DB Free Tier
Essentials

Status
Online
Resource group (move)
iiot-book-2-resource
Subscription (move)
Azure subscription 1
Subscription ID
ecf0e257-9639-462c-a6e5-ad307cb4241e
Total throughput limit
1000 RU/s

Figure 12.6 – Adding a new container to Cosmos DB

Then, provide a database name and a container name and set the partition key to `/deviceId`. See the following figure for reference:



* Database id ⓘ

● Create new ○ Use existing

iiotdata

☑ Share throughput across containers ⓘ

* Database throughput (autoscale) ⓘ

● Autoscale ○ Manual

Estimate your required RU/s with capacity calculator.

Database Max RU/s ⓘ

100 *

Your database throughput will automatically scale from **10 RU/s (10% of max RU/s) - 100 RU/s** based on usage.

Estimated monthly cost (USD) ⓘ: **$0.88 - $8.76** (1 region, 10
100 RU/s, $0.00012/RU)

* Container id ⓘ

iiotdata-container

* Indexing

● Automatic ○ Off

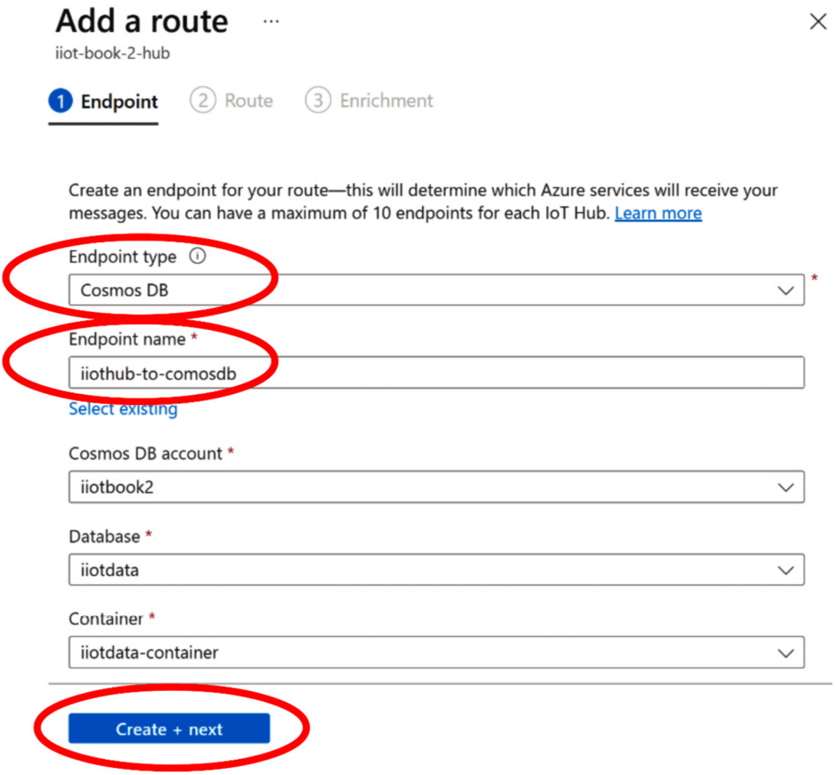
All properties in your documents will be indexed by default f
flexible and efficient queries. Learn more

* Partition key ⓘ

/deviceId

Figure 12.7 – Creating a Cosmos DB container

Now that Cosmos DB is up and running, the next step is to configure a route on IoT Hub to send messages to Cosmos DB. From IoT Hub, add a route to Cosmos DB, as shown in the next figure, then create it:



Add a route ...
iiot-book-2-hub
1 Endpoint
2 Route
3 Enrichment

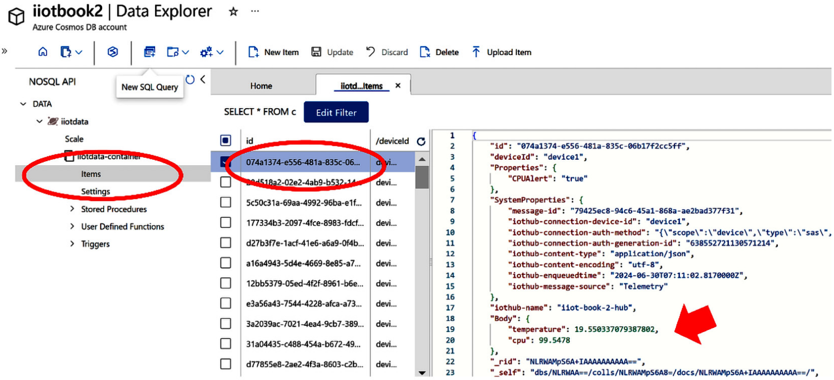
Create an endpoint for your route—this will determine which Azure services will receive your messages. You can have a maximum of 10 endpoints for each IoT Hub. Learn more

Endpoint type: Cosmos DB *
Endpoint name *: iiothub-to-comosdb
Cosmos DB account *: iiotbook2
Database *: iiotdata
Container *: iiotdata-container

Select existing
Create + next

Figure 12.8 – Creating the IoT Hub route

We have completed our integration between IoT Hub and Cosmos DB. If everything goes well, when we restart our edge device, we will see the data flow to Cosmos DB. Click on Cosmos DB | Data Explorer | Items and you will see the data in Cosmos DB, as shown here:


iiotbook2 | Data Explorer
Azure Cosmos DB account
New Item
Update
Discard
Delete
Upload Item
NOSQL API
New SQL Query
Home
iiotd...Items
DATA
iiotdata
Scale
iiotdata-container
Items
Settings
Stored Procedures
User Defined Functions
Triggers
Edit Filter

SELECT * FROM c

id	/deviceId
074a1374-e556-481a-835c-06...	devi...
20d518a2-02e2-4ab9-b532-14...	devi...
5c50c31a-69aa-4992-96ba-e1f...	devi...
177334b3-2097-4fce-8983-fdcf...	devi...
d27b3f7e-1acf-41e6-a6a9-0f4b...	devi...
a16a4943-5d4e-4669-8e85-a7...	devi...
12bb5379-05ed-4f2f-8961-b6e...	devi...
e3a56a43-7544-4228-afca-a73...	devi...
3a2039ac-7021-4ea4-9cb7-389...	devi...
31a04435-c488-454a-b672-49...	devi...
d77855e8-2ae2-4f3a-8603-c2b...	devi...

```
{
    "id": "074a1374-e556-481a-835c-06b17f2cc5ff",
    "deviceId": "device1",
    "Properties": {
        "CPUAlert": "true"
    },
    "SystemProperties": {
        "message-id": "79425ec8-94c6-45a1-868a-ae2bad377f31",
        "iothub-connection-device-id": "device1",
        "iothub-connection-auth-method": "{\"scope\":\"device\",\"type\":\"sas\",
        "iothub-connection-auth-generation-id": "638552721130571214",
        "iothub-content-type": "application/json",
        "iothub-content-encoding": "utf-8",
        "iothub-enqueuedtime": "2024-06-30T07:11:02.8170000Z",
        "iothub-message-source": "Telemetry"
    },
    "iothub-name": "iiot-book-2-hub",
    "Body": {
        "temperature": 19.550337079387802,
        "cpu": 99.5478
    },
    "_rid": "NLRWAMpS6A+IAAAAAAAAAA==",
    "_self": "dbs/NLRWAA==/colls/NLRWAMpS6A8=/docs/NLRWAMpS6A+IAAAAAAAAAA==/",
```

Figure 12.9 – Showing the data in Cosmos DB

Well done, we have our data in the cloud ready to be viewed or processed. It is important to define a specific partitioning key for our use case; a partition key allows Cosmos DB to distribute and organize the data equally between the partitions and share the workload fairly. Therefore, a good partition key improves performance.

In this case, we accepted the one proposed by Azure based on a combination of device ID and time, which is the most common use case. Normally, we analyze the latest data sent for each edge or group of edges; however, if our use case is different, we need to rethink this partition.

Time Series Insights

Time Series Insights (TSI) is a database that supports time series data. The Time Series Insights service was retired on 7 July 2024, but Azure offers migration support for your data (https://learn.microsoft.com/en-us/azure/time-series-insights/how-to-tsi-gen2-migration).

Azure offers many SQL and NoSQL data storage services. IIoT use cases are better suited to NoSQL. Azure suggests Cosmos DB, but you can also leverage third-party databases.

In this section, we learned how to save our measures data on Azure. In the next subsections, we will see how to process it.

Data processing

Azure provides three different frameworks that can be used to process data: Stream Analytics, Data Lake Analytics, and ML analytics. In our exercise, we will use Azure Stream Analytics to process data *online* (called a Hot path).

Azure Stream Analytics

Azure Stream Analytics (ASA) allows building simple analytics such as excursion monitoring analytics (EMA); see *Chapter 7* . The key element of ASA is the query language called ASQL , which is based on SQL and has additional built-in functions. The most common features of ASQL are as follows:

- Time management : ASQL provides functions to work with event time
- Windowing : ASQL supports several windowing functions to handle time event data streaming
- Aggregation : Like SQL, ASQL defines aggregation functions
- User-defined functions : ASQL allows developers to write their own custom functions

- Input and output operators : ASQL allows us to pre- or post-process data on the fly

Azure Stream Analytics offers built-in machine learning-based anomaly detection capabilities through two special functions:

- **`AnomalyDetection_SpikeAndDip`** : This function detects temporary anomalies such as spikes or dips in a time series event stream
- **`AnomalyDetection_ChangePoint`** : This detects anomalies such as changes in the distribution of values, level changes, and trends in a time series stream

These functions allow performing anomaly detection directly within your Stream Analytics job, without the need for external machine learning models.

Working with Stream Analytics

In this example, we will extract the efficiency of the simulated device and save the results into Table storage, but before developing our analytics, we need to start our device. We are going to reuse the

Python edge device developed before (refer to *Chapter 11*). From the command line, run the following command:

```
docker compose -f .\docker-compose-
python.yml up
```

Now, we need to build ASA jobs. To do so, we need to search for ASA using the search bar (see *Figure 12 .10*). Once you have found it, click on Create | Stream Analytics jobs . These steps are shown in the following screenshot:

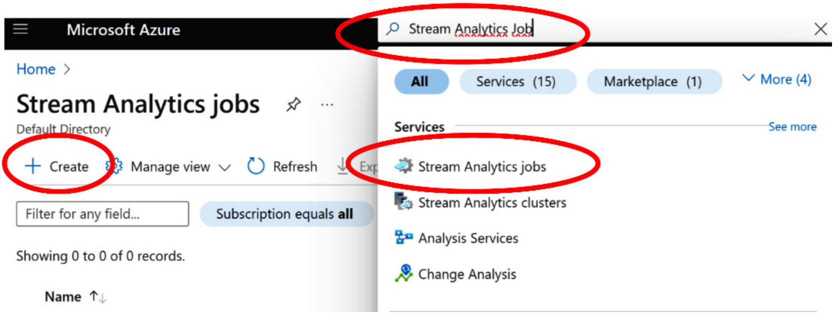
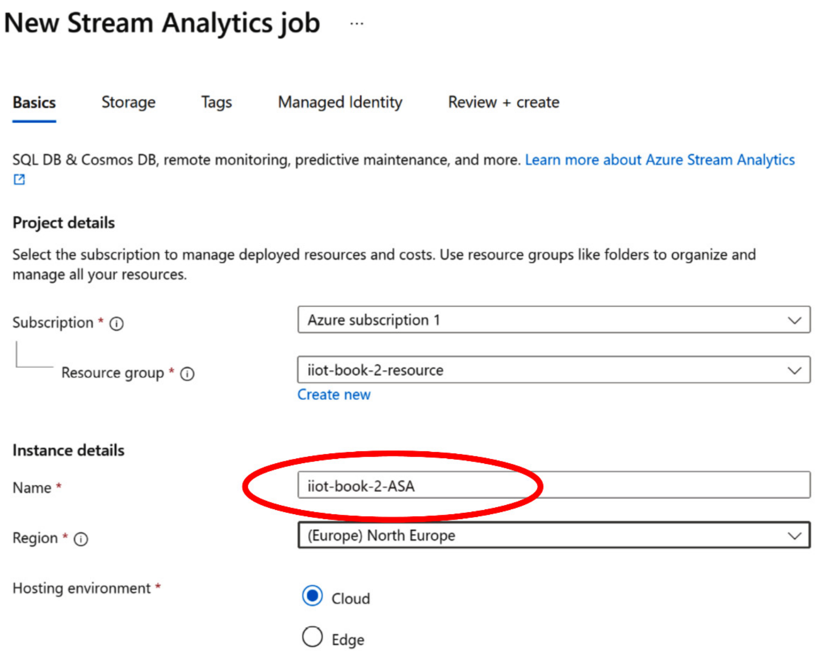

Microsoft Azure
Stream Analytics Job
Home >
Stream Analytics jobs
Default Directory
+ Create
Manage view
Refresh
Exp
Filter for any field...
Subscription equals all
Showing 0 to 0 of 0 records.
Name
All
Services (15)
Marketplace (1)
More (4)
Services
See more
Stream Analytics jobs
Stream Analytics clusters
Analysis Services
Change Analysis

Figure 12.10 – Creating ASA jobs

In the next step, provide a name for your ASA jobs, as shown here:


New Stream Analytics job ···
Basics
Storage
Tags
Managed Identity
Review + create
Learn more about Azure Stream Analytics
Create new

New Stream Analytics job

SQL DB & Cosmos DB, remote monitoring, predictive maintenance, and more. Learn more about Azure Stream Analytics

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription *: Azure subscription 1

Resource group *: iiot-book-2-resource

Instance details

Name *: iiot-book-2-ASA

Region *: (Europe) North Europe

Hosting environment *: ● Cloud ○ Edge

Figure 12.11 – Starting ASA jobs on the cloud

Finally, set the streaming units to 3 and create the ASA job.

Streaming units allow us to define the number of jobs for our data processing. For our exercise, *3* is enough.

Once ASA is deployed, we need to define an input. From the left menu, click on Inputs and add IoT Hub , as shown in the following figure:

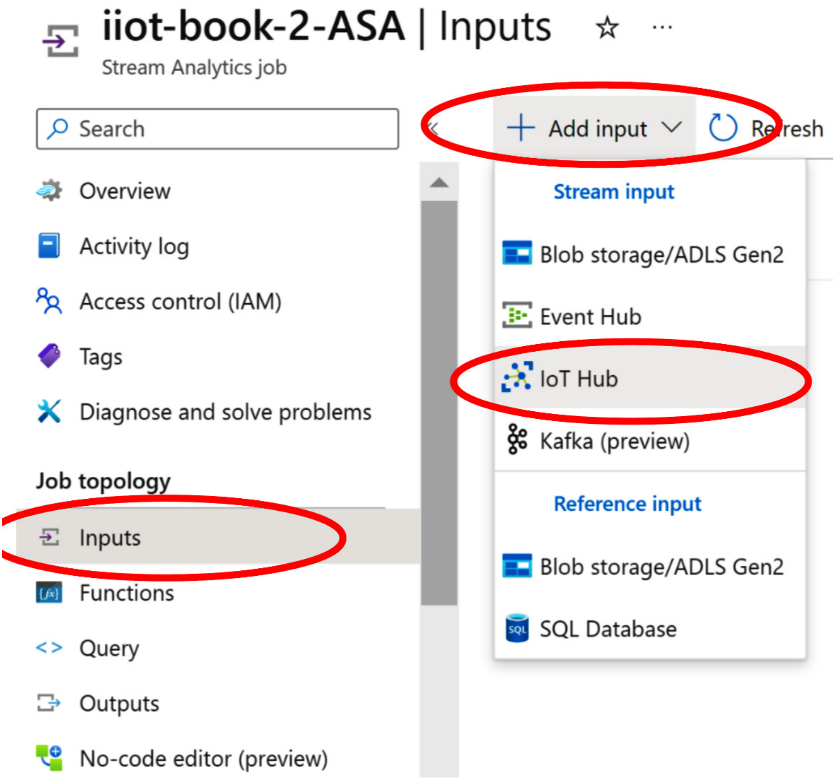
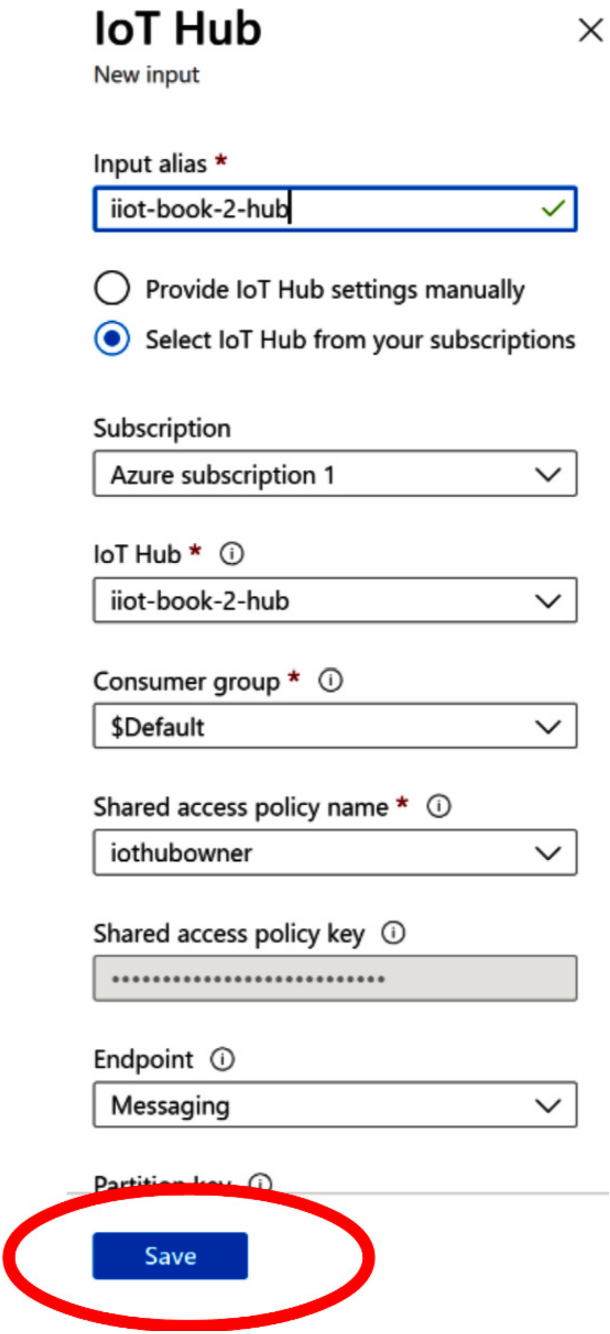

iiot-book-2-ASA | Inputs
Stream Analytics job
Search
Overview
Activity log
Access control (IAM)
Tags
Diagnose and solve problems
Job topology
Inputs
Functions
Query
Outputs
No-code editor (preview)
Add input
Refresh
Stream input
Blob storage/ADLS Gen2
Event Hub
IoT Hub
Kafka (preview)
Reference input
Blob storage/ADLS Gen2
SQL Database

Figure 12.12 – Adding IoT Hub as input

Finally, accept all the default parameters, as shown here:



IoT Hub

New input

Input alias * iiot-book-2-hub

○ Provide IoT Hub settings manually

● Select IoT Hub from your subscriptions

Subscription: Azure subscription 1

IoT Hub *: iiot-book-2-hub

Consumer group *: $Default

Shared access policy name *: iothubowner

Shared access policy key: ••••••••••••••••••••••••••••

Endpoint: Messaging

Figure 12.13 – Saving IoT Hub as input

Similarly, we can define the output of the ASA. From the left menu, click on Outputs and add Table storage , as shown here:

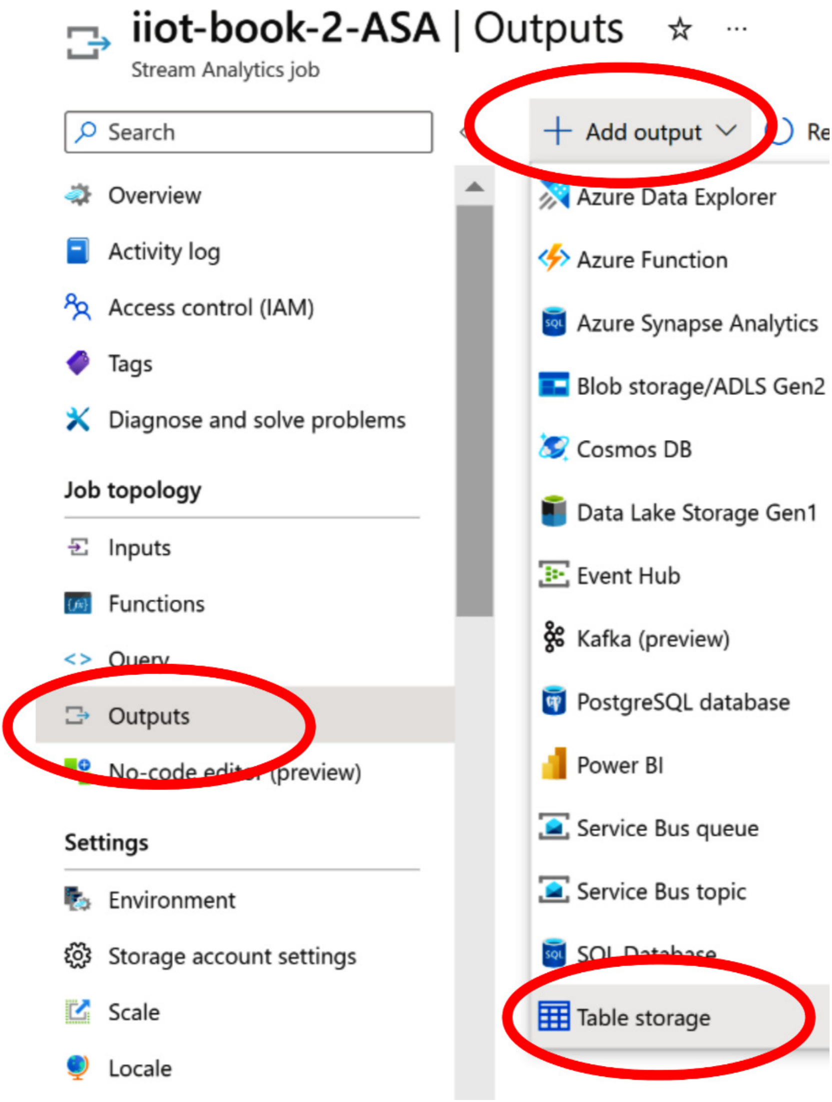

iiot-book-2-ASA | Outputs
Stream Analytics job
Search
Overview
Activity log
Access control (IAM)
Tags
Diagnose and solve problems
Job topology
Inputs
Functions
Query
Outputs
No-code editor (preview)
Settings
Environment
Storage account settings
Scale
Locale
Add output
Re
Azure Data Explorer
Azure Function
Azure Synapse Analytics
Blob storage/ADLS Gen2
Cosmos DB
Data Lake Storage Gen1
Event Hub
Kafka (preview)
PostgreSQL database
Power BI
Service Bus queue
Service Bus topic
SQL Database
Table storage

Figure 12.14 – Adding Table storage as a new

output

Finally, provide a name (in our case, `iiotbook2asaout`) and provide the name of the table, the partition key, and the row key. For Partition key , we used `deviceId` , and for Row key , we used `messageId` . See the following figure for reference:

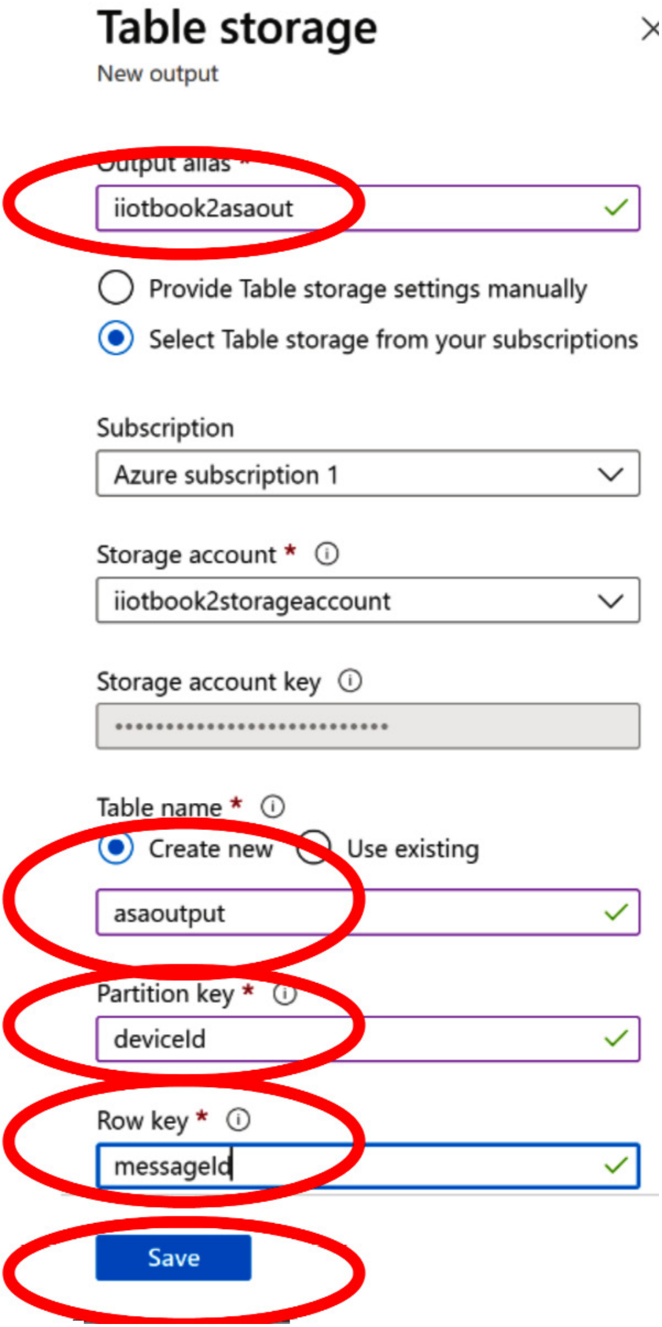


Table storage

New output

Output alias *

iiotbook2asaout

○ Provide Table storage settings manually

● Select Table storage from your subscriptions

Subscription

Azure subscription 1

Storage account *

iiotbook2storageaccount

Storage account key

••••••••••••••••••••••••••••••

Table name *

● Create new ○ Use existing

asaoutput

Partition key *

deviceId

Row key *

messageId

Figure 12.15 – Setting the Table storage for the

ASA output

Having added the source input to Azure IoT Hub and the output, the next step is to develop an ASA job to process our data. From the left menu, click on Query and define the following query:

```
SELECT
    IoTHub.MessageId as messageId,
IoTHub.ConnectionDeviceId as deviceId,
cpu, temperature as tempC
INTO
    [iiotbook2asaout]
FROM
    [iiot-book-2-hub]
WHERE cpu >100
```

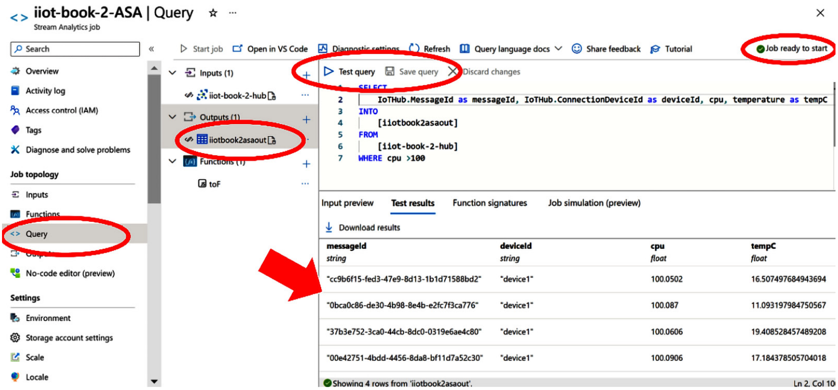
The previous query extracts only the message with a CPU greater than 100 and stores **`deviceId`** , **`messageId`** , **`cpu`** , and **`temperature`** in the output table defined before.

> ### *Partition key*

> *Please note that we previously defined* **`deviceId`** *as the partition key;*

> *it is important to define a partition key to allow Azure to store data efficiently.*

Now, click on Test query to see the output. The following figure shows the steps to do this and the expected output:


iiot-book-2-ASA | Query
Stream Analytics job
Search
Overview
Activity log
Access control (IAM)
Tags
Diagnose and solve problems
Job topology
Inputs
Functions
Query
No-code editor (preview)
Settings
Environment
Storage account settings
Scale
Locale
Start job
Open in VS Code
Diagnostic settings
Refresh
Query language docs
Share feedback
Tutorial
Job ready to start
Inputs (1)
iiot-book-2-hub
Outputs (1)
iiotbook2asaout
Functions (1)
toF
Test query
Save query
Discard changes
Input preview
Test results
Function signatures
Job simulation (preview)
Download results
Showing 4 rows from 'iiotbook2asaout'.
Ln 2, Col 10

```
SELECT
    IoTHub.MessageId as messageId, IoTHub.ConnectionDeviceId as deviceId, cpu, temperature as tempC
INTO
    [iiotbook2asaout]
FROM
    [iiot-book-2-hub]
WHERE cpu >100
```

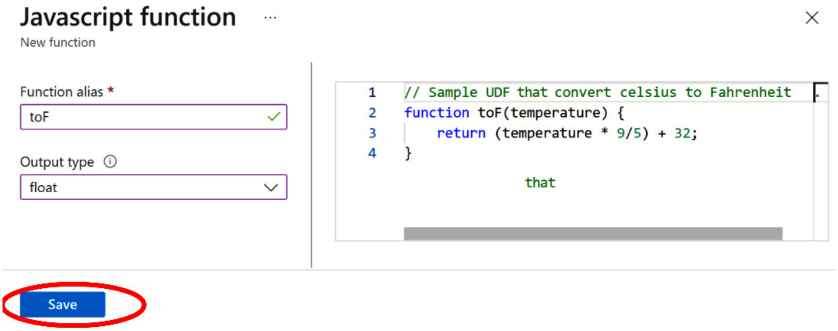
messageId string	deviceId string	cpu float	tempC float
"cc9b6f15-fed3-47e9-8d13-1b1d71588bd2"	"device1"	100.0502	16.507497684943694
"0bca0c86-de30-4b98-8e4b-e2fc7f3ca776"	"device1"	100.087	11.093197984750567
"37b3e752-3ca0-44cb-8dc0-0319e6ae4c80"	"device1"	100.0606	19.408528457489208
"00e42751-4bdd-4456-8da8-bf11d7a52c30"	"device1"	100.0906	17.184378505704018

Figure 12.16 – Defining a new query

User-defined function on ASA

Let us increase the complexity level of our job. ASA allows us to define a custom function; to explore this functionality, we will create a user-defined function (UDF) to convert Celsius to Fahrenheit.

To create this simple UDF to convert data, we can click on the Functions tab in the menu on the left, and create a JavaScript function called `toF` with the following code:



Function alias * toF

Output type float

```
// Sample UDF that convert celsius to Fahrenheit
function toF(temperature) {
    return (temperature * 9/5) + 32;
}
```

Figure 12.17 – Defining a custom function

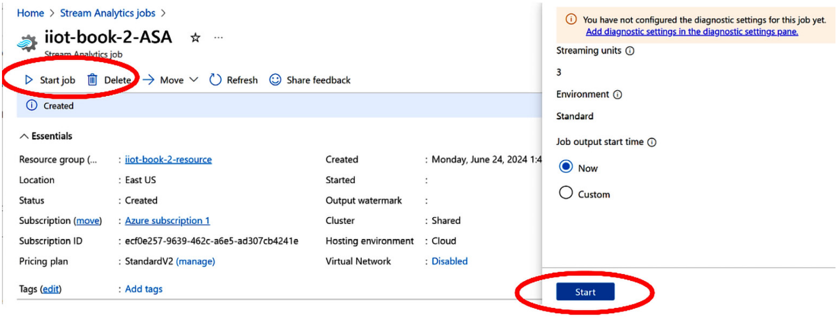
For ease of reference, the code of our function can also be found here:

```
// Sample UDF that converts Celsius to
Fahrenheit .
function toF(temperature) {
    return (temperature * 9/5) + 32;
}
```

Then, refine the query as shown in the following figure and save it:

```
SELECT
    IoTHub.MessageId as messageId,
IoTHub.ConnectionDeviceId as deviceId,
cpu, temperature as tempC,
udf.toF(temperature) as tempF
INTO
    [iiotbook2asaout]
FROM
    [iiot-book-2-hub]
WHERE cpu >100
```

Finally, remember to start the job by clicking on Start from the Overview tab. This is shown in the following screenshot:


Home > Stream Analytics jobs >
iiot-book-2-ASA
Stream Analytics job
Start job
Delete
Move
Refresh
Share feedback
Created
Essentials
Start

You have not configured the diagnostic settings for this job yet. Add diagnostic settings in the diagnostic settings pane.

Resource group (... : iiot-book-2-resource
Location : East US
Status : Created
Subscription (move) : Azure subscription 1
Subscription ID : ecf0e257-9639-462c-a6e5-ad307cb4241e
Pricing plan : StandardV2 (manage)
Tags (edit) : Add tags
Created : Monday, June 24, 2024 1:4
Started :
Output watermark :
Cluster : Shared
Hosting environment : Cloud
Virtual Network : Disabled

Streaming units
3
Environment
Standard
Job output start time
● Now
○ Custom

Figure 12.18 – Starting the ASA job

> ## *Stop ASA*

> *Do not forget to stop the ASA job just created to not incur undesired costs. From ASA jobs, click on the just created job and click on* ***Stop job*** *. Azure will start the job immediately.*

To see the output, search for `Storage Browser` in the Azure portal, as shown here:

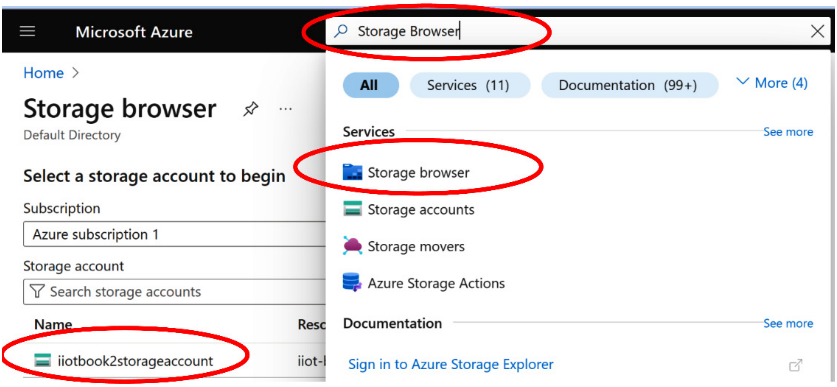
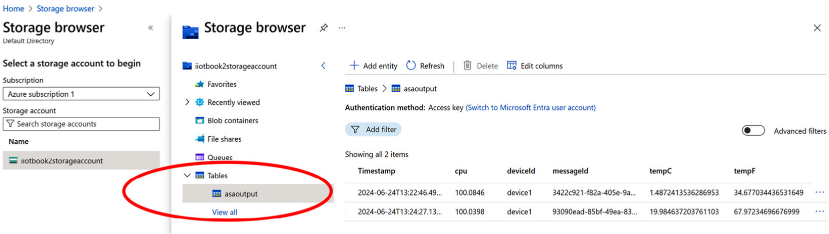

Microsoft Azure
Storage Browser
Home >
Storage browser
Default Directory
Select a storage account to begin
Subscription
Azure subscription 1
Storage account
Search storage accounts
Name
Resc
iiotbook2storageaccount
iiot-
All
Services (11)
Documentation (99+)
More (4)
Services
See more
Storage browser
Storage accounts
Storage movers
Azure Storage Actions
Documentation
See more
Sign in to Azure Storage Explorer

Figure 12.19 – Locating the storage browser in the Azure portal

Then, open the table to see the saved message. As you can see, only the events with a CPU greater than 100 have been saved:


Home > Storage browser >
Storage browser
Default Directory
Select a storage account to begin
Subscription
Azure subscription 1
Storage account
Search storage accounts
Name
iiotbook2storageaccount
Storage browser
iiotbook2storageaccount
Favorites
Recently viewed
Blob containers
File shares
Queues
Tables
asaoutput
View all
Add entity
Refresh
Delete
Edit columns
Tables > asaoutput
Add filter
Advanced filters

Authentication method: Access key (Switch to Microsoft Entra user account)

Showing all 2 items

Timestamp	cpu	deviceId	messageId	tempC	tempF
2024-06-24T13:22:46.49...	100.0846	device1	3422c921-f82a-405e-9a...	1.4872413536286953	34.677034436531649
2024-06-24T13:24:27.13...	100.0398	device1	93090ead-85bf-49ea-83...	19.984637203761103	67.97234696676999

Figure 12.20 – Exploring the ASA output

Advanced Stream Analytics

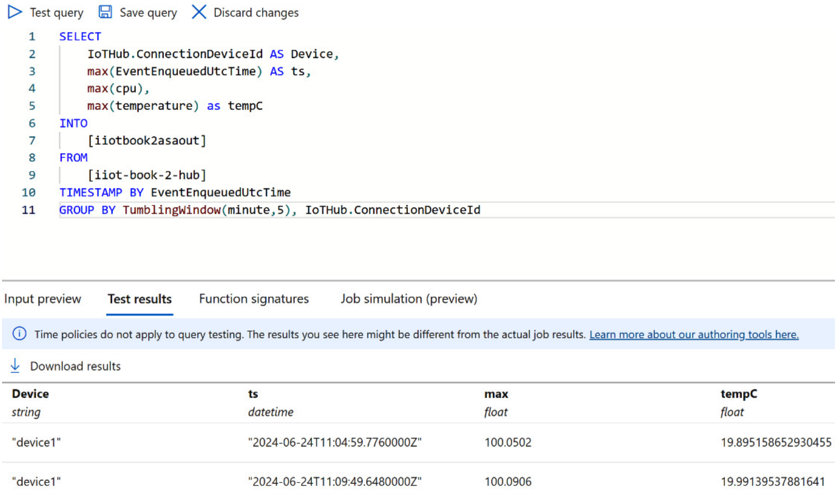
In the previous section, we discovered how Stream

Analytics works. In this section, we will discuss some of the more advanced features of this service, particularly the query language. The Query tab allows us to define a simple, SQL-like query to process data and store it in a storage system. This mechanism can also be applied to different storage mechanisms such as a SQL database, Event Hubs, Blob Storage, Power BI, and Azure Functions. We can also link different input data sources, such as other data streams or reference data. Reference data is statically stored, normally in a blob store, that can be joined with the input data stream. An example of reference data would be the asset information.

The advanced Stream Analytics query language also allows you to group by time using a method called windowing , which means that you can define a relative time interval, such as five minutes, to aggregate measures. The following query extracts the maximum CPU and maximum temperature from the last five minutes:

```
SELECT
    IoTHub.ConnectionDeviceId AS
Device,
    max(EventEnqueuedUtcTime) AS ts,
    max(cpu),
    max(temperature) as tempC
INTO
    [iiotbook2asaout]
FROM
    [iiot-book-2-hub]
TIMESTAMP BY EventEnqueuedUtcTime
GROUP BY TumblingWindow(minute,5),
IoTHub.ConnectionDeviceId
```

The following screenshot shows the expected output when the data is sampled every five minutes. It is also important to group by **`deviceId`** to avoid mixing data between devices:


Test query
Save query
Discard changes

```
SELECT
    IoTHub.ConnectionDeviceId AS Device,
    max(EventEnqueuedUtcTime) AS ts,
    max(cpu),
    max(temperature) as tempC
INTO
    [iiotbook2asaout]
FROM
    [iiot-book-2-hub]
TIMESTAMP BY EventEnqueuedUtcTime
GROUP BY TumblingWindow(minute,5), IoTHub.ConnectionDeviceId
```

Input preview
Test results
Function signatures
Job simulation (preview)

Time policies do not apply to query testing. The results you see here might be different from the actual job results. Learn more about our authoring tools here.

Download results

Device *string*	**ts** *datetime*	**max** *float*	**tempC** *float*
"device1"	"2024-06-24T11:04:59.7760000Z"	100.0502	19.895158652930455
"device1"	"2024-06-24T11:09:49.6480000Z"	100.0906	19.99139537881641

Figure 12.21 – Testing the Stream Analytics query language

Azure supports 3 types of windowed grouping, 14 aggregate functions, and a very powerful time management system for skipping out-of-order data (https://learn.microsoft.com/en-us/stream-analytics-query/stream-analytics-query-language-reference).

Out-of-order data

Out-of-order data is data that arrived too late to be processed. Sometimes, events arrive late due to connectivity issues, latency, or bottlenecks. This can

> *cause strange artifacts in data processing. Analytics are the most vulnerable to these issues as they make decisions not only on the specific data but also on the context (i.e., the history of the data previously sent). If this data is out of order, the analytics cannot use it or will use it incorrectly. We will study these effects in Chapter 13 . Azure allows developers to set up policy orders. You can learn more at https://learn.microsoft.com/en-us/azure/stream-analytics/event-ordering .*

Azure Synapse

Azure Synapse is a new service for near-real-time analytics and big data analytics. With Azure Synapse, we can process big data using advanced SQL and/or Spark Notebook or simply visualize data.

Despite Azure Synapse not being specific for IoT data, we can leverage it to perform query and big data analytics on our raw data. Indeed, Synapse is fully integrated with Cosmos DB.

Prerequisites for Azure Synapse

To work with Azure Synapse, we need to enable a

Synapse subscription. So, from the Subscription menu, enable a Synapse subscription, as shown here:

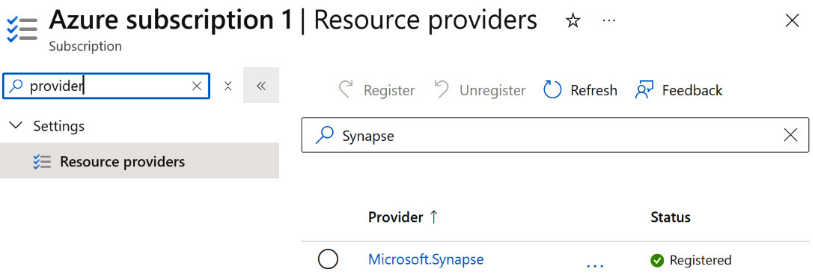

Azure subscription 1 | Resource providers
Subscription
provider
Settings
Resource providers
Register
Unregister
Refresh
Feedback
Synapse
Provider ↑
Status
Microsoft.Synapse
Registered

Figure 12.22 – Enabling an Azure Synapse subscription

Creating an Azure Synapse workspace

Now, we can create an Azure Synapse workspace. A workspace is an environment where we can build our analytics, SQL, Spark job, and data transformation pipeline.

From the Azure portal, search for `Synapse` and create a new workspace:

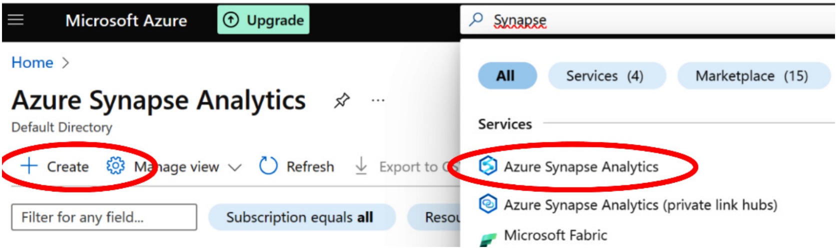
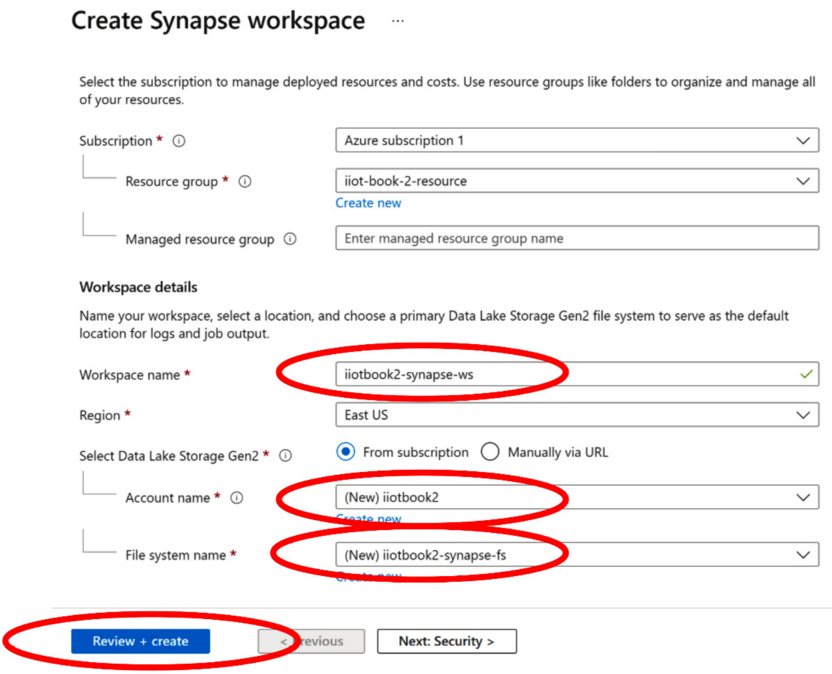

Microsoft Azure
Upgrade
Synapse
Home >
Azure Synapse Analytics
Default Directory
+ Create
Manage view
Refresh
Export to
Filter for any field...
Subscription equals all
Resou
All
Services (4)
Marketplace (15)
Services
Azure Synapse Analytics
Azure Synapse Analytics (private link hubs)
Microsoft Fabric

Figure 12.23 – Creating an Azure Synapse workspace

Then, provide names for Workspace name , Account name , and File system name , as shown in the following figure:


Create Synapse workspace ...

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all of your resources.

Subscription * : Azure subscription 1
Resource group * : iiot-book-2-resource
Managed resource group : Enter managed resource group name

Workspace details

Name your workspace, select a location, and choose a primary Data Lake Storage Gen2 file system to serve as the default location for logs and job output.

Workspace name * : iiotbook2-synapse-ws
Region * : East US
Select Data Lake Storage Gen2 * : ● From subscription ○ Manually via URL
Account name * : (New) iiotbook2
File system name * : (New) iiotbook2-synapse-fs

Create new
Create new
Create new
Review + create
< Previous
Next: Security >

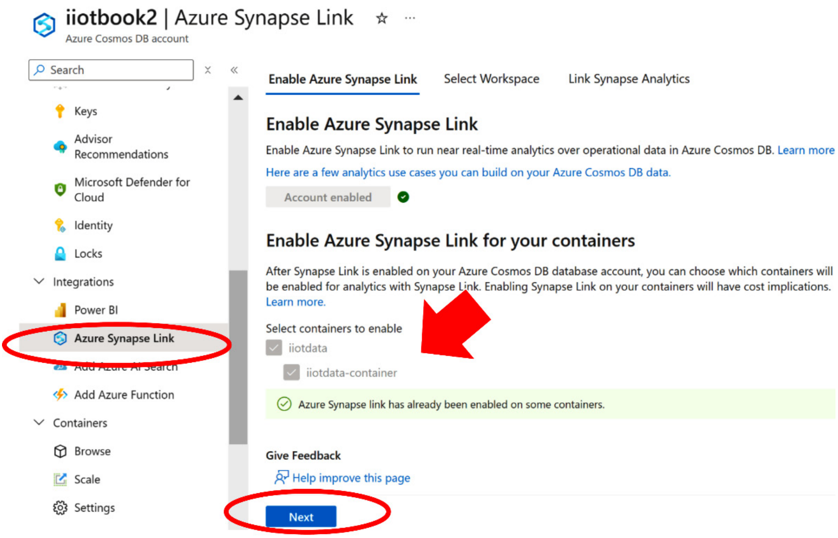
Figure 12.24 – Setting up an Azure Synapse

workspace

Working with Azure Synapse and Cosmos DB

When the workspace is ready, we must link our data source to Synapse. In our exercise, we want to visualize data ingested from the edge device with the temperature of our simulated boiler above 19°C.

Access the Cosmos DB created before and, from the left menu, select Azure Synapse Link , and enable the Synapse Link as shown in the following figure:


iiotbook2 | Azure Synapse Link
Azure Cosmos DB account
Search
Keys
Advisor Recommendations
Microsoft Defender for Cloud
Identity
Locks
Integrations
Power BI
Azure Synapse Link
Add Azure AI Search
Add Azure Function
Containers
Browse
Scale
Settings
Enable Azure Synapse Link
Select Workspace
Link Synapse Analytics
Account enabled
Give Feedback
Help improve this page
Next

Enable Azure Synapse Link

Enable Azure Synapse Link to run near real-time analytics over operational data in Azure Cosmos DB. Learn more

Here are a few analytics use cases you can build on your Azure Cosmos DB data.

Enable Azure Synapse Link for your containers

After Synapse Link is enabled on your Azure Cosmos DB database account, you can choose which containers will be enabled for analytics with Synapse Link. Enabling Synapse Link on your containers will have cost implications. Learn more.

Select containers to enable

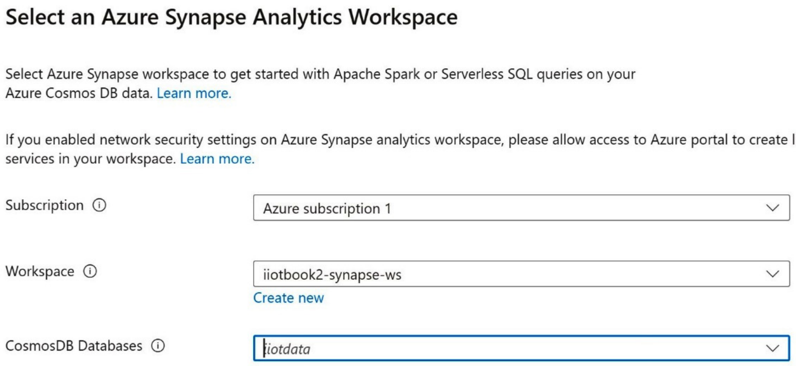
☑ iiotdata

☑ iiotdata-container

Azure Synapse link has already been enabled on some containers.

Figure 12.25 – Enabling Cosmos DB to work with

In the next step, link Cosmos DB with the Synapse workspace, as shown in the following figure:



Select an Azure Synapse Analytics Workspace

Select Azure Synapse workspace to get started with Apache Spark or Serverless SQL queries on your Azure Cosmos DB data. Learn more.

If you enabled network security settings on Azure Synapse analytics workspace, please allow access to Azure portal to create linked services in your workspace. Learn more.

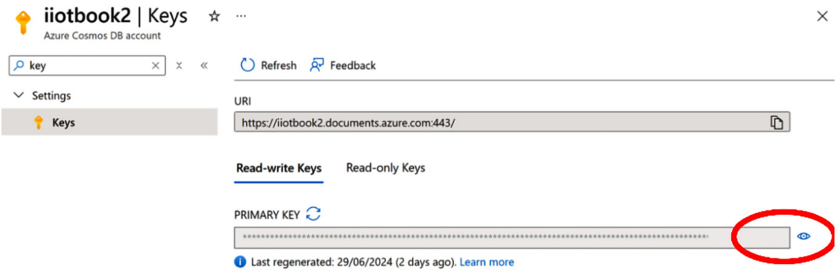
Subscription: Azure subscription 1

Workspace: iiotbook2-synapse-ws

CosmosDB Databases: iotdata

Figure 12.26 – Connecting the Synapse workspace to Cosmos DB

Now, we are ready to work with Synapse Studio, but we first need to carry out one additional step. We need to get the read-write keys from Cosmos DB. From the left menu, go to Keys , then copy the primary key, as shown in the following figure:


iiotbook2 | Keys
Azure Cosmos DB account
key
Settings
Keys
Refresh
Feedback
Read-write Keys
Read-only Keys
Last regenerated: 29/06/2024 (2 days ago). Learn more

URI

https://iiotbook2.documents.azure.com:443/

PRIMARY KEY

Figure 12.27 – Getting the keys to authenticate the Synapse workspace

Indeed, we need to authenticate Synapse to connect to Cosmos DB.

The final step is to open Synapse Studio from the Synapse workspace.

Click on the data source and, in the Cosmos DB database just linked, open a new SQL script, and authenticate to Cosmos DB:

```
CREATE CREDENTIAL [iiotbook2]
WITH IDENTITY = 'SHARED ACCESS
SIGNATURE', SECRET = '<your key>'
```

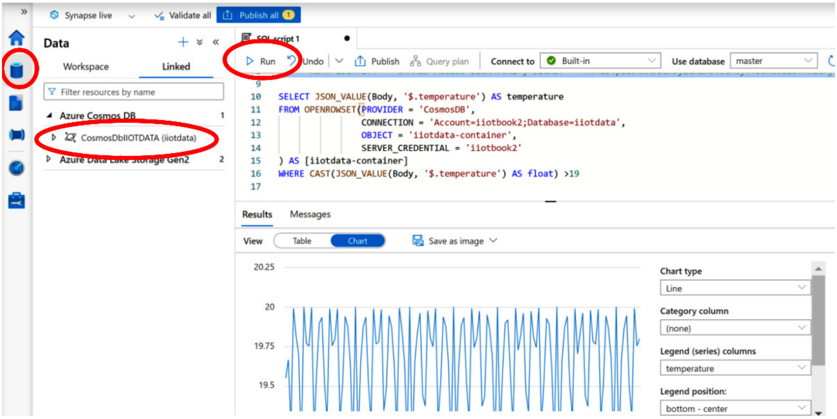
Here, `your key` is the key copied before.

Now, we are ready to perform our analytical query:

```
SELECT JSON_VALUE(Body,
'$.temperature') AS temperature
FROM OPENROWSET( PROVIDER =
'CosmosDB',
                CONNECTION =
'Account=iiotbook2;Database=iiotdata',
                OBJECT = 'iiotdata-
container',
                SERVER_CREDENTIAL =
'iiotbook2'
) AS [iiotdata-container]
WHERE CAST(JSON_VALUE(Body,
'$.temperature') AS float) >19
```

Please note the **`JSON_VALUE`** function used in the query. Synapse supports various built-in JSON functions to extract data from the message body.

The expected outcome is shown here:


Synapse live
Validate all
Publish all 1
Data
Workspace
Linked
Filter resources by name
Azure Cosmos DB 1
CosmosDbIIOTDATA (iiotdata)
Azure Data Lake Storage Gen2 2
SQL script 1
Run
Undo
Publish
Query plan
Results
Messages
View
Table
Chart
Save as image
20.25
20
19.75
19.5

Connect to: Built-in

Use database: master

```
SELECT JSON_VALUE(Body, '$.temperature') AS temperature
FROM OPENROWSET(PROVIDER = 'CosmosDB',
                CONNECTION = 'Account=iiotbook2;Database=iiotdata',
                OBJECT = 'iiotdata-container',
                SERVER_CREDENTIAL = 'iiotbook2'
) AS [iiotdata-container]
WHERE CAST(JSON_VALUE(Body, '$.temperature') AS float) >19
```

Chart type: Line

Category column: (none)

Legend (series) columns: temperature

Legend position: bottom - center

Figure 12.28 – Showing the temperature data

from Cosmos DB

Well done! We connected Azure Synapse with Cosmos DB to start a simple analytic job. With Synapse Studio, you can build your pipeline to connect analytics to your live data and schedule it recurrently. From the left menu, click on the Pipeline menu and add a new pipeline. Now, you can build on several components such as ML analytics, Batch services, Azure Data Explorer, and so on (see also https://learn.microsoft.com/en-us/azure/data-factory/how-to-create-schedule-trigger?tabs=synapse-analytics).

Machine Learning analytics

Azure Machine Learning (ML) analytics is an environment in which we can deploy advanced analytics for predictive modeling. ML analytics can be used also from a Synapse workspace.

In this section, we configured Cosmos DB and

Synapse to work together and built stream analytics with Azure Stream Analytics.

We will learn more about analytics and ML in *Chapters 13* , *14* , and *15* . In the next section, we will explore how to visualize data for users.

Building visualizations

Visualization is the last component of our architecture. Through visualization applications, users can monitor the status of assets and alarms and build powerful dashboards. Azure offers several options for data visualization:

- Azure Data Explorer
- Managed Grafana
- Power BI
- Azure IoT Central

We have already seen Grafana in *Chapters 8* and *10* . In a later subsection, we will see the basic steps to configure it.

Azure Data Explorer

Azure Data Explorer is a managed service to efficiently handle data. Azure Data Explorer allows us to process, export, and visualize data. Data can be locally stored or simply connected through connectors. Connectors support IoT Hub, Cosmos DB, Event Hubs, and Event Grid. Moreover, we can use an ingestion SDK to connect different data sources.

Finally, we can manage data efficiently using a powerful query language called Kusto Query Language (KQL).

To work with Azure Data Explorer, search for it in the Azure portal search bar, as shown in the following figure, and create a new cluster:

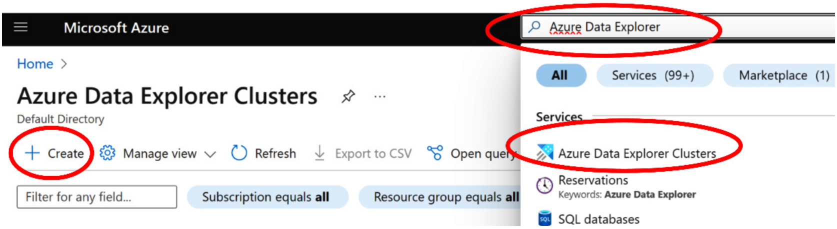
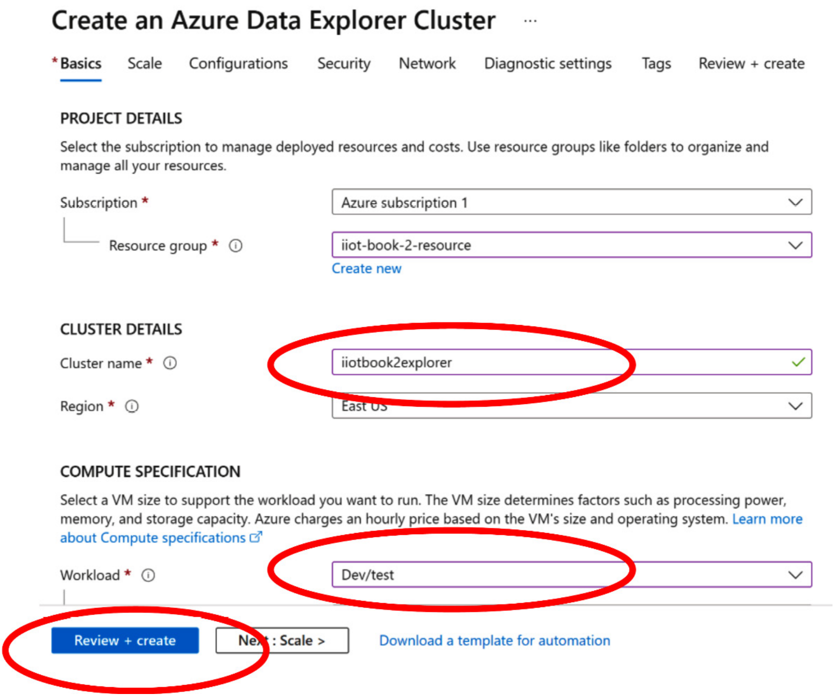

Microsoft Azure
Azure Data Explorer
Home >
Azure Data Explorer Clusters
Default Directory
+ Create
Manage view
Refresh
Export to CSV
Open quer
Filter for any field...
Subscription equals all
Resource group equals all
All
Services (99+)
Marketplace (1)
Services
Azure Data Explorer Clusters
Reservations
Keywords: Azure Data Explorer
SQL databases

Figure 12.29 – Locating Azure Data Explorer

Configure the cluster by providing a cluster name (for instance, `iiotbook2explorer`) and setting Workload to Dev/test . See the following figure:


Create an Azure Data Explorer Cluster ...
*Basics
Scale
Configurations
Security
Network
Diagnostic settings
Tags
Review + create
Create new
Review + create
Next : Scale >
Download a template for automation

PROJECT DETAILS

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * Azure subscription 1

Resource group * iiot-book-2-resource

CLUSTER DETAILS

Cluster name * iiotbook2explorer

Region * East US

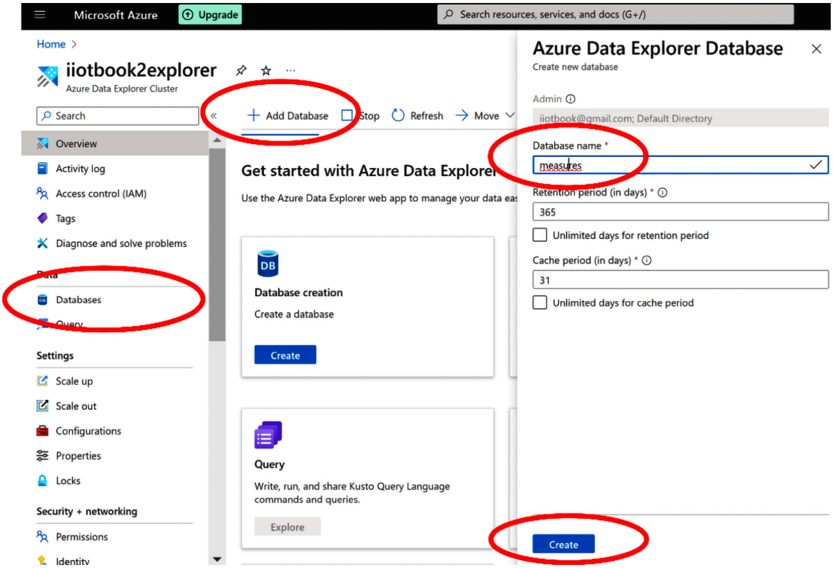
COMPUTE SPECIFICATION

Select a VM size to support the workload you want to run. The VM size determines factors such as processing power, memory, and storage capacity. Azure charges an hourly price based on the VM's size and operating system. Learn more about Compute specifications

Workload * Dev/test

Figure 12.30 – Creating an Azure Data Explorer cluster

Now, we need to create a database. From the left menu, click on Databases | Add Database and provide a database name, as shown in the following figure:


Microsoft Azure
Upgrade
Search resources, services, and docs (G+/)
Home >
iiotbook2explorer
Azure Data Explorer Cluster
Search
+ Add Database
Stop
Refresh
→ Move
Overview
Activity log
Access control (IAM)
Tags
Diagnose and solve problems
Data
Databases
Query
Settings
Scale up
Scale out
Configurations
Properties
Locks
Security + networking
Permissions
Identity
Get started with Azure Data Explorer
Use the Azure Data Explorer web app to manage your data eas
Database creation
Create a database
Create
Query
Write, run, and share Kusto Query Language commands and queries.
Explore
Azure Data Explorer Database
Create new database
Create

Admin: iiotbook@gmail.com; Default Directory

Database name *: measures

Retention period (in days) *: 365

☐ Unlimited days for retention period

Cache period (in days) *: 31

☐ Unlimited days for cache period

Figure 12.31 – Creating a database in Azure Data Explorer

We do not want to store data, but for our exercise, we can simply connect to external data sources.

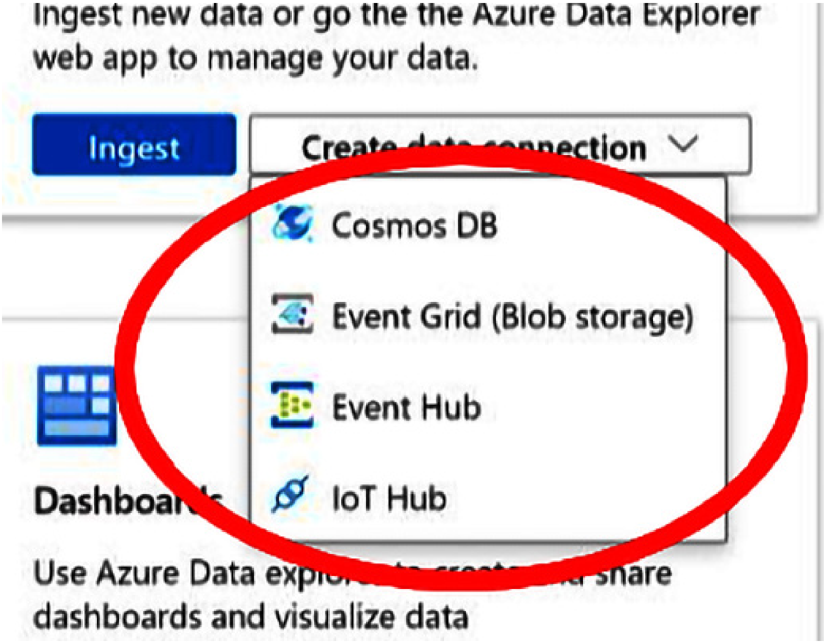
In the just-created database, create a table with the following fields:

- **`deviceid as string`**
- **`cpu and temperature as real`**
- **`timestamp as datetime`**

Finally, create a mapping called **`JsonMapping`** to map Cosmos DB to our database. From the left menu, click on Query and create a new JSON mapping. Use the following query to create the mapping:

```
.create table data ingestion json
mapping "JsonMapping"
'['
'{ "column" : "deviceid", "datatype" :
"string", "Properties":
{"Path":"$.deviceid"}},'
'{ "column" : "temperature",
"datatype" : "real", "Properties":
{"Path":"$.Body.temperature"}},'
'{ "column" : "cpu", "datatype" :
"real", "Properties":
{"Path":"$.Body.cpu"}},'
'{ "column" : "timestamp", "datatype"
: "datetime", "Properties":
{"Path":"$.timestamp"}},'
']'
```

From Azure Data Explorer | Overview , we can ingest data from Cosmos DB. From the Ingest menu, create a connection to Cosmos DB. See the following figure:



Ingest new data or go the the Azure Data Explorer web app to manage your data.

Ingest
Create data connection
Cosmos DB
Event Grid (Blob storage)
Event Hub
IoT Hub
Dashboards

Use Azure Data explorer to create and share dashboards and visualize data

Figure 12.32 – Connecting to Cosmos DB

On the next screen, select the Cosmos DB database and set the `JsonMapping` mapping we just created.

Now, you are ready to work with KQL to perform your analysis. We will use Azure Data Explorer instance just created with Managed Grafana.

Grafana

Azure supports Grafana, which we have already encountered in *Chapters 8* and *10* . To work with

Grafana, search for `Azure Managed Grafana` in the search bar and configure the plugin to access the Azure Data Explorer instance created in the previous paragraph.

From the Azure portal, search for `Grafana` and create a new instance, as shown in the following figure:

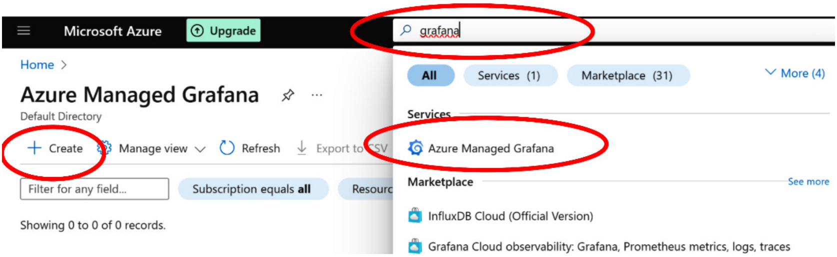
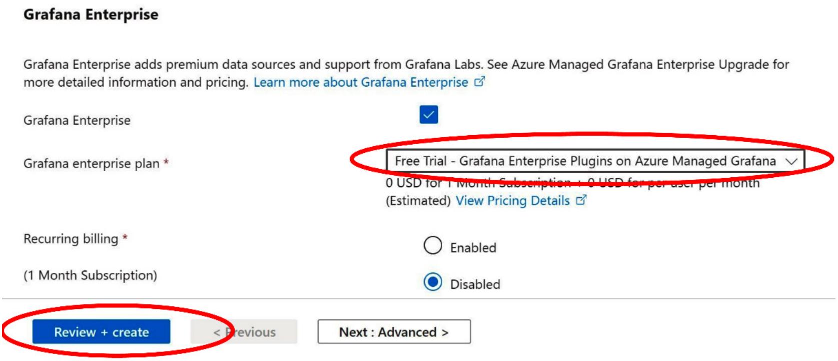

Microsoft Azure
Upgrade
grafana
Home >
Azure Managed Grafana
Default Directory
+ Create
Manage view
Refresh
Export to CSV
Filter for any field...
Subscription equals all
Resourc
Showing 0 to 0 of 0 records.
All
Services (1)
Marketplace (31)
More (4)
Services
Azure Managed Grafana
Marketplace
See more
InfluxDB Cloud (Official Version)
Grafana Cloud observability: Grafana, Prometheus metrics, logs, traces

Figure 12.33 – Creating a Grafana service

Choose an *Enterprise* license because we need advanced plugins. You can see the following figure for reference:



Grafana Enterprise

Grafana Enterprise adds premium data sources and support from Grafana Labs. See Azure Managed Grafana Enterprise Upgrade for more detailed information and pricing. Learn more about Grafana Enterprise

Grafana Enterprise ☑

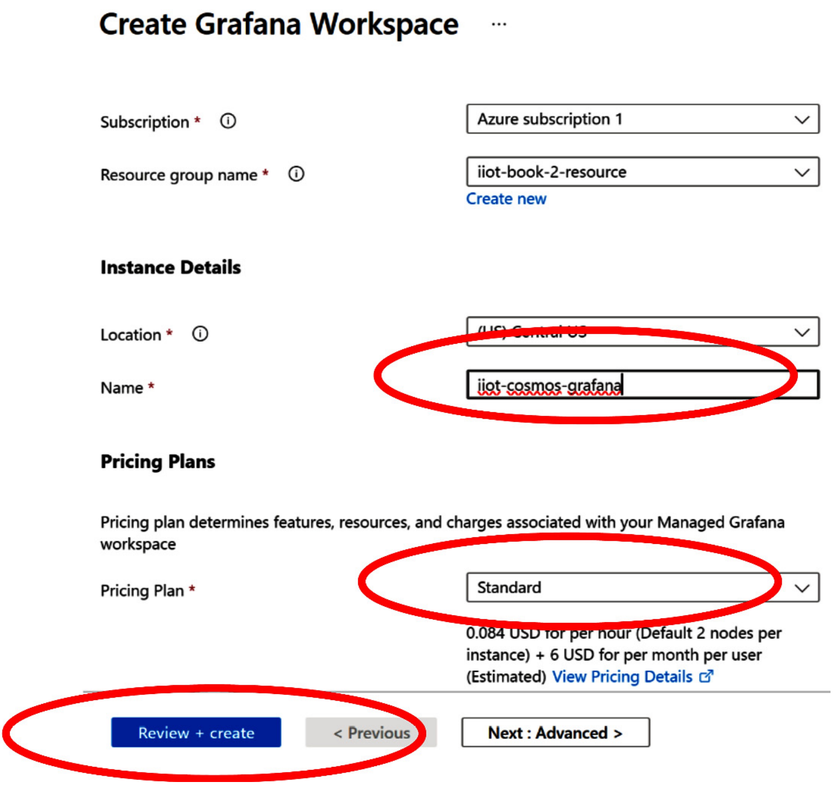
Grafana enterprise plan * Free Trial - Grafana Enterprise Plugins on Azure Managed Grafana
0 USD for 1 Month Subscription + 0 USD for per user per month (Estimated) View Pricing Details

Recurring billing * ○ Enabled

(1 Month Subscription) ● Disabled

Figure 12.34 – Configuring a new instance of Grafana

Finally, create a new workspace, as shown here:



Create Grafana Workspace ...

Subscription * Azure subscription 1

Resource group name * iiot-book-2-resource
Create new

Instance Details

Location * (US) Central US

Name * iiot-cosmos-grafana

Pricing Plans

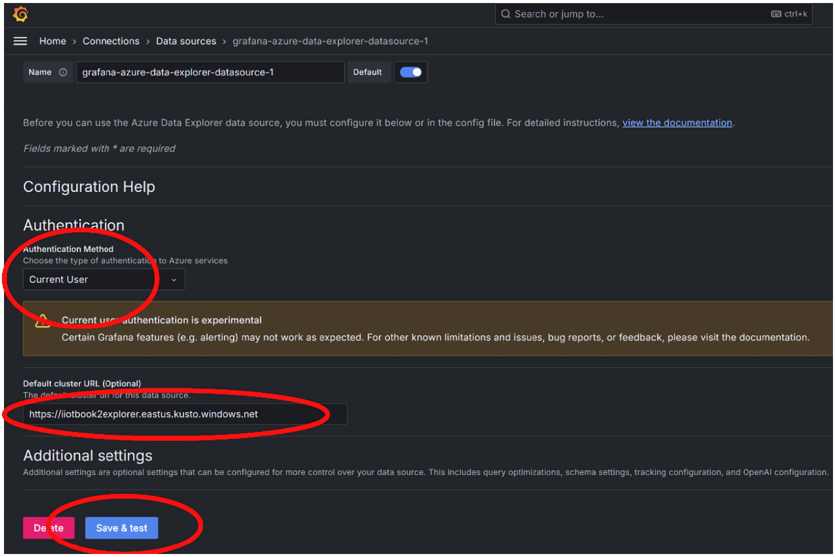
Pricing plan determines features, resources, and charges associated with your Managed Grafana workspace

Pricing Plan * Standard

0.084 USD for per hour (Default 2 nodes per instance) + 6 USD for per month per user (Estimated) View Pricing Details

Figure 12.35 – Creating a Grafana workspace

We are now ready to connect our data source. From Overview , click on the Grafana URL, and from the left menu, add a new data source, as shown in the following figure. For the URL, copy and paste the URL of the Azure Data Explorer instance created in the previous section:


Search or jump to...
ctrl+k
Home › Connections › Data sources › grafana-azure-data-explorer-datasource-1

Name: grafana-azure-data-explorer-datasource-1

Default: ●

Before you can use the Azure Data Explorer data source, you must configure it below or in the config file. For detailed instructions, view the documentation.

*Fields marked with * are required*

Configuration Help

Authentication

Authentication Method

Choose the type of authentication to Azure services

Current User

Current user authentication is experimental

Certain Grafana features (e.g. alerting) may not work as expected. For other known limitations and issues, bug reports, or feedback, please visit the documentation.

Default cluster URL (Optional)

The default cluster url for this data source.

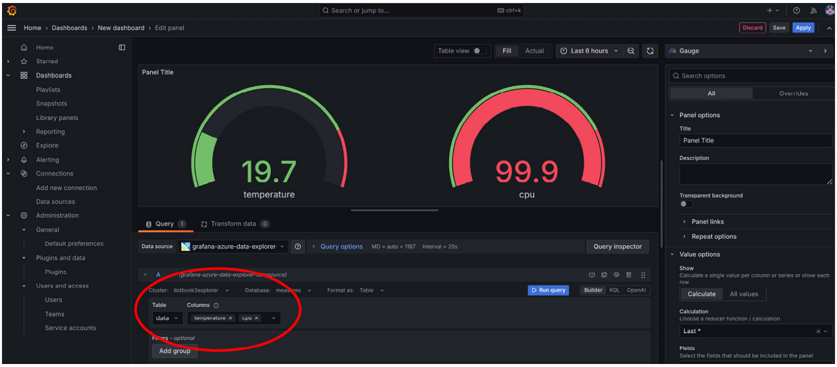
https://iiotbook2explorer.eastus.kusto.windows.net

Additional settings

Additional settings are optional settings that can be configured for more control over your data source. This includes query optimizations, schema settings, tracking configuration, and OpenAI configuration.

Figure 12.36 – Connecting the data source

Finally, develop your dashboard to your liking. We have already seen how to do it in *Chapters 8* and *10* . The following figure shows the final step:


Search or jump to...
ctrl+k
Home › Dashboards › New dashboard › Edit panel
Discard
Save
Apply
Home
Starred
Dashboards
Playlists
Snapshots
Library panels
Reporting
Explore
Alerting
Connections
Add new connection
Data sources
Administration
General
Default preferences
Plugins and data
Plugins
Users and access
Users
Teams
Service accounts
Table view
Fill
Actual
Last 6 hours
Panel Title
19.7
temperature
99.9
cpu
Query 1
Transform data 0
Query options
Query inspector
Run query
Builder
KQL
OpenAI
Filters - optional
Add group
Gauge
Search options
All
Overrides
Panel options
Panel links
Repeat options
Value options
Calculate
All values

Data source: grafana-azure-data-explorer

MD = auto = 1187 Interval = 20s

A (grafana-azure-data-explorer-datasource)

Cluster: iiotbook2explorer

Database: measures

Format as: Table

Table: data

Columns: temperature, cpu

Title: Panel Title

Description

Transparent background

Show: Calculate a single value per column or series or show each row

Calculation: Choose a reducer function / calculation: Last *

Fields: Select the fields that should be included in the panel

Figure 12.37 – Configuring the Grafana dashboard

Power BI

Power BI is a suite for business analytics that provides insights and allows you to see live data. With Power BI, we can build a live dashboard with just a few clicks.

You can visit https://learn.microsoft.com/en-us/azure/cosmos-db/nosql/powerbi-visualize to connect Power BI and Cosmos DB.

Another simple alternative to Power BI is the IoT Central application.

IoT Central

IoT Central is a new service that can connect IoT Edge and IoT Hub with a few clicks. IoT Central is a PaaS application to manage devices at scale. As we know by now, IIoT is different from IoT. Indeed, IoT must manage multiple devices rather than complex plants. The IoT Central application seems perfect for IoT but you should limit the customization of the platform.

You can visit https://azure.microsoft.com/en-us/products/iot-central to learn more.

More on Azure Data Explorer

Azure Data Explorer is an external service supported by Azure. Azure Data Explorer connects to Event Hubs or IoT Hub to read data and allows you to easily develop dashboards to monitor sensors. To work with Azure Data Explorer, search for the service in the Azure portal and create a cluster. Then, connect Azure Data Explorer to IoT Hub and build your custom dashboard.

You can visit https://dataexplorer.azure.com to learn more.

In this section, we have studied several visualization tools and have concluded our architecture. Soon, interfaces will change drastically, becoming conversational human-machine interaction systems. In a few years, AI will probably help us to *chat* with our data in a more interactive way. For this reason, we suggest developing your application based on your needs.

Comparing the platforms

Let's conclude our discussion by comparing the discovered platforms. To evaluate a platform, it is important to take the specific use case into account, but there are three general key points that we should consider:

- Does the data model conform to how we acquire data?
- Does the support for analytics conform to our needs?
- Are the supported devices suitable for our use cases?

In IIoT, we acquire data when it changes, so we may need interpolation functions to fill the gaps, as we discovered with the InfluxDB open source databases in *Chapter 8* . It is important to consider whether, for our context, it is more appropriate to use an analytical Cold path or a Hot path. Microsoft has good support for the OPC DA and OPC UA acquisition standards, while AWS is currently developing support for OPC UA on Greengrass.

With regard to the key points mentioned here, it seems that while Azure is a very mature platform for IIoT, AWS is filling the gap to integrate standard protocols and OPC UA on Greengrass, and GCP performs strongly with regard to processing data. U-SQL and the windowing mechanism of ASA seem to be very powerful in the context of IIoT as well.

From a cost perspective, which of the three platforms is more suitable will strictly depend on the discount plans on offer. In general, however, Microsoft Azure has the lowest price, while AWS usually comes in the middle.

Summary

In this chapter, we explored the most common functionalities of Azure to build an IIoT data flow. With this, we have come to the end of our discussion on data acquisition technologies. In the next chapter, we will explore how to develop analytics in the industrial sector. We will learn about diagnostic, maintenance, and predictive analytics, and discover both physics-based and data-driven technologies. We will develop several examples using ML techniques and deploy our analytics on the cloud.

Part 3:Industrial Artificial Intelligence of Things

This final section helps the reader grasp the concepts of analytics and artificial intelligence in an industrial context. Through hands-on experience, readers will learn to develop algorithms for anomaly detection, predictive maintenance, machine learning, and digital twin applications.

This part includes the following chapters:

- *Chapter 13* , *Performing Diagnostic, Maintenance, and Predictive Analytics*
- *Chapter 14* , *Implementing a Digital Twin – Advanced Analytics*
- *Chapter 15* , *Deploying an Analytics Model*

13

Performing Diagnostic, Maintenance, and Predictive Analytics

In the previous chapter, we implemented Industrial Internet of Things (IIoT) platforms based on open source, Amazon Web Services (AWS), and Azure.

In this chapter, we will discover the most important use cases for IIoT analytics. We will see the main classes of analytics and their applications. We will study, step by step, what the main steps to take to develop analytics are. We will understand the OSA architecture. Finally, we will put into practice what we have learned by developing three different types of analytics in Python. In particular, we will explore the following topics:

- IIoT analytics
- The different classes of analytics
- IIoT analytics technologies
- Building IIoT analytics
- Deploying analytics
- Analytics in practice
- Machine Learning (ML) analytics

Technical requirements

In this chapter, we need the following prerequisites:

- Git: https://git-scm.com/downloads
- Python 3.9+: https://www.python.org/downloads
- Docker: https://www.docker.com/products/docker-desktop/

The code for this chapter is available at the official repository at https://github.com/PacktPublishing/Hands-On-Industrial-Internet-of-Things-Second-Edition .

Jupyter

If you have installed Python, you may want to use a development IDE. One of the best IDE is the Jupyter Notebook. To install Jupyter, run the following commands:

```
$ pip install notebook scikit-learn
numpy pandas
```

To run the notebook, run the following command from the command line:

```
$ jupyter notebook
```

If you want to avoid installing Python and Anaconda, you can also run Jupyter Notebook by starting the `jupyter/datascience-notebook` container.

Start the container as follows:

```
$ docker run -p 8888:8888 -v ./work:/
work -i --name jupyterhub-ds jupyter/
datascience-notebook
```

Then, connect the browser to `http://localhost:8888/?token=<provided token>`.

Open a new notebook for coding.

IIoT analytics

Analytics are one of the most important aspects of the IIoT and are, without a doubt, the most complex. To manage complex systems, we need advanced techniques. Since these technologies are very advanced, they can provide valuable and reliable insights, reducing false positives and false negatives. We can think, for example, of the analytics applied to avionic systems or those that apply in safety contexts; clearly, we need high reliability. Over the last five years, the authors of this book have collaborated to develop more than 150 different analytics and thousands of simple rules that can be applied to thousands of different assets. The following are the most important use cases of these analytics that we have identified in the IIoT:

- Asset reliability management : Without a doubt, this is the most common class of analytics. It is used to monitor the health of the system. An example of where this may be applied is checking the standard operability of a pump or predicting cracking or malfunctioning.

- Maintenance optimization/planning : This is the most valuable class. Its purpose is to avoid or delay unwanted maintenance. An example of where this may be applied is avoiding unwanted cleaning or part replacements of a filter.
- Performance and efficiency management or optimization : These analytics try to minimize the cost while maintaining performance. An example of where this may be applied is optimizing oil extraction.
- Regulatory and compliance : Sometimes, we need to monitor Key Performance Indicators (KPIs) to ensure that production remains within the constraints of regional laws. An example of where this may be applied is to regulate sulfur emissions at a refinery.
- Operation and scheduling optimization : This isn't specifically related to IIoT, but it is needed to manage standard operations. An example of where this may be applied is to optimize the maintenance schedule of a train of assets.
- Security and safety : These are rules to check the safety or security of humans or machines. An example of where this may be applied is to measure the reliability of a jet engine.
- Design and modeling : These are normally used during the design phase. They allow you to redesign the production. An example of where this may be applied is to optimize piping.
- What-if simulation for decision support : These are normally applied to human interactions. These techniques allow you to change the parameters of a particular analytic to simulate the behavior of

equipment such as wind turbines, oil and gas plants, or equipment related to the pharmaceutical industry. An example of where this may be applied is changing the furnace temperature in a petrochemical process.

- Smart environment : This involves monitoring large environments or utilities. An example of where this may be applied is to control human access to prohibited areas.

All of these technologies can be applied to the IIoT in conjunction with the standard supply chain, Enterprise Resource Planning (ERP), Business Intelligence (BI), or process optimization.

In this section, we have introduced the most important use cases on IIoT. In the next chapter, we will introduce the most important type of analytics from a technology point of view.

The different classes of analytics

Analytics can be applied for different purposes and using different techniques. They are generally separated into four different families: descriptive, diagnostic, predictive, and prescriptive. These are ordered from the simplest to the most advanced.

Descriptive analytics

Descriptive analytics is the most basic class of analytics. They analyze past data to try and provide a broad view of the fleet. In other words, descriptive analytics tries to answer the question, " *What happened?* ". These analytics use data mining, aggregation, or visual intelligence techniques to understand the status of the assets. In the IIoT, the most common technique used for descriptive analytics is KPI monitoring.

KPI and health monitoring

KPI monitoring is the simplest way to monitor the health of a fleet by aggregating or calculating new indicators from raw data. Moreover, we can implement a condition monitoring mechanism using a specific simple rule.

Condition monitoring

In *Chapters 7* , *8* , *10* , and *12* , we developed a few basic analytics based on thresholding. Condition monitoring is exactly this kind of thing. It uses thresholds or fuzzy rules to extract potential issues that occurred in the past. These analytics can be developed using simple rules, clustering algorithms, simple moving averages, or more advanced analytics such as the Kalman filter.

Diagnostic analytics

Diagnostic analytics are the most common analytics in the IIoT. These analytics use advanced modeling techniques to analyze failure modes and extract the root cause of an issue. Diagnostic analytics try to reply to the question " *Why did it happen* ?".

The three steps of diagnostic analytics are as follows:

1. Detect anomalies.
2. Discover anomalies.

3. Determine the root cause of anomalies.

Normally, the last two steps of diagnostic analytics are done by human investigation. Diagnostic analytics can use feature extraction or anomaly reasoners to provide indicators about why an anomaly happened. Anomaly detection is, on the contrary, an automatic step performed by (more or less) sophisticated analytics.

Anomaly detection

Anomaly detection algorithms have been applied to the IIoT since its induction. Anomaly detection analytics is a special class of descriptive analytics to catch and sometimes anticipate anomalies from a piece of equipment.

Anomaly detection analytics is a special extension of descriptive analytics used to identify and sometimes predict unusual patterns in data from a particular piece of equipment. By analyzing data, anomaly detection algorithms would flag deviations from normal behavior that may indicate a problem or impending failure. This allows for proactive maintenance.

We can distinguish between three types of anomaly detection, like so:

- Point anomaly : A single data point is anomalous if it's too far from the expected behavior
- Collective anomalies : A group of data points helps to detect the anomalies
- Contextual anomalies : Anomalies are inherently dependent on the specific context in which they occur

These three types are shown in the following diagram:

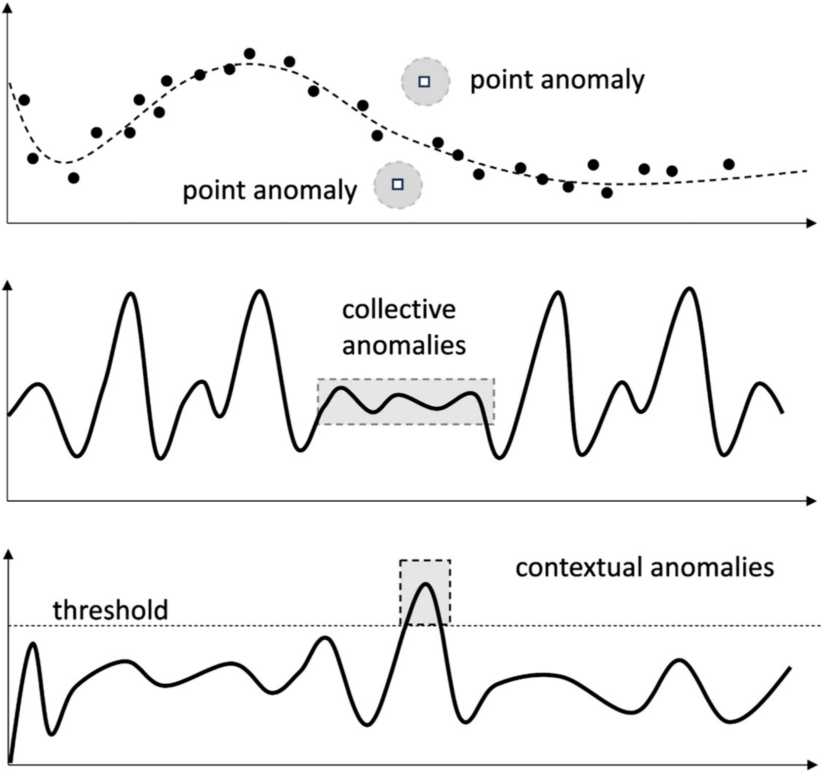

point anomaly
point anomaly
collective
anomalies
contextual anomalies
threshold

Figure 13.1 – The three different types of anomaly detection

Generally, these analytics build a data-driven model based on the equipment's standard operability by monitoring it. This is then compared to the current data or features. If the difference is too high, an alert is raised to inform the user of a possible malfunction.

These analytics can be developed using simple rules, clustering algorithms, simple moving averages, or more advanced analytics such as autoencoder, diffusion models, and deep learning models in general.

Discovering anomalies and the root cause of anomalies

After anomaly detection, we need to discover the failure mode and identify the cause of the issue. Normally, these activities require human knowledge of failure modes and effect analysis or a large dataset of past failures. For instance, we can implement a set of rules (which can be deterministic, fuzzy, Bayesian, or ML-based), codifying the cause and effect of the fault.

Predictive analytics

Predictive analytics look at the future and attempt to answer the question “ *What could happen in the* *future* ?”. These analytics use regression models or anomaly detection models to anticipate potential issues that might occur in the future.

Prognostic analytics

In the IIoT sector, we need to pay special attention to prognostic analytics . Prognostic analytics is an estimation of time to failure and risk for one or more existing and future failure modes (ISO 13381-1: 2015). Prognostic analytics predict the future degradation or damage and the Remaining Useful Life (RUL) of an asset or part based on the measured data.

The goal of prognostics is to predict the cycles remaining before the damage grows beyond the eligibility threshold. To predict the RUL, prognostics utilize the measured damage levels or an estimation of the damage up to the current cycle.

Unfortunately, these measures are often affected by uncertainty, which might include noise, bad measurements, and lack of knowledge. When we estimate the RUL, we also have to quantify the Probability Density Function (PDF) of how accurate the prediction is. The process of estimating these uncertainties is called Uncertainty Quantification (UQ). The following diagram summarizes these concepts:

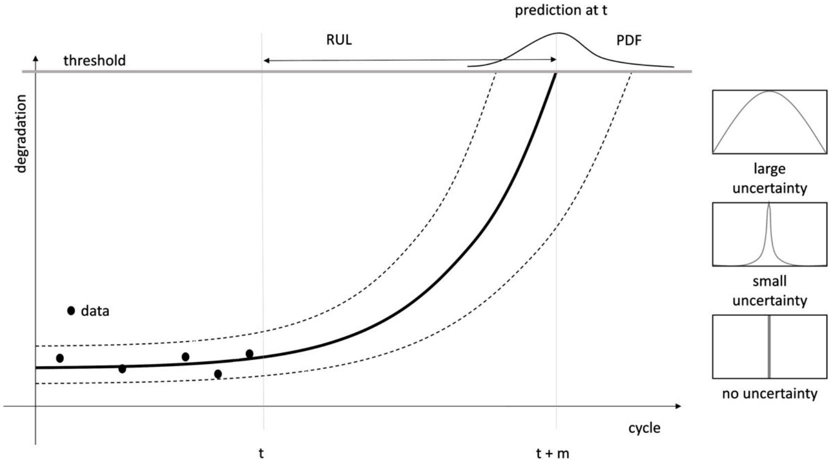

prediction at t
RUL
PDF
threshold
degradation
● data
large uncertainty
small uncertainty
no uncertainty
cycle
t
t + m

Figure 13.2 – Degradation prediction in a prognostic model, including the RUL and the UQ (adaptation from Prognostics and Health Management of Engineering Systems, Nam-Ho Kim, Dawn An, and Joo-Ho Choi, Springer 2017)

The most interesting way to develop prognostic analytics is by developing a digital twin model. We will see a practical implementation in *Chapter 14* .

Other predictive analytics

Predictive analytics are mainly used to predict variables exogenous to the system to make business or maintenance decisions in advance, for example, by predicting the market demand you can plan production (see the *Predictive production analytic in practice* section), machine load, or plan maintenance. More simply, the forecast of atmospheric events helps to define the maintenance intervals of energy plants.

Prescriptive analytics

Prescriptive analytics represents the next step in this prediction process and asks the question " *How should we respond to these potential future events* ?". Prescriptive analytics anticipates what will happen, when it will happen, and why, suggesting different options related to decisions. These analytics forecast future opportunities and/or mitigate future risks. The common application of prescriptive analytics in IIoT is for Condition-Based Maintenance (CBM).

CBM

CBM is a maintenance strategy for repairing or replacing damaged or degraded parts that might reduce the life expectancy of a machine. It monitors, detects, isolates, and predicts equipment performance and degradation without shutting down daily production. In CBM, the maintenance of systems and components is based on the health of the equipment versus the possibility it might break down or require scheduled maintenance. The following diagram shows the basic idea of CBM compared to more standard maintenance approaches:

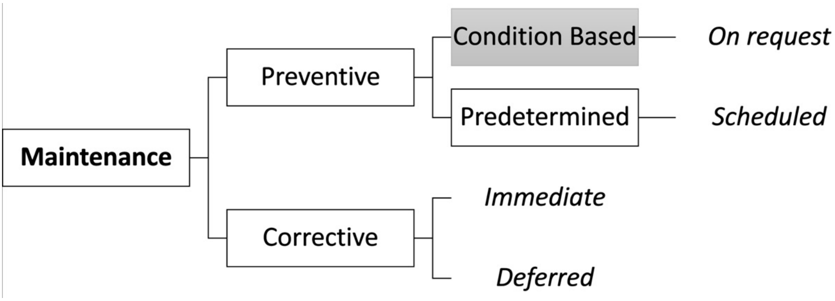

Maintenance
Preventive
Corrective
Condition Based
Predetermined
On request
Scheduled
Immediate
Deferred

Figure 13.3 – The scope of work of the CBM

CBM tries to answer the question " *Should we change this part or piece of equipment, in accordance with our maintenance plan, or shall we take the risk and continue operating* ?".

Production optimization analytics

Optimization analytics is an interesting class of analytics that is related to IIoT. They try to maximize efficiency by increasing production and reducing costs. These analytics work together with data and are normally either activated proactively by a user or used daily to provide automatic insight to the user.

The industrial sector has used optimization analytics on-premises since 1990 to optimize production directly from the controller. These analytics are not properly part of the IIoT, but they should be considered in our discussion.

In this section, we analyzed the different classes of IIoT analytics from the point of view of their purpose (*WHAT*). In the next section we will look at them from the point of view of how they do it (*HOW*).

IIoT analytics technologies

From a technical point of view, IIoT analytics uses multiple modules—ML, anomaly detection, physics kernels, risk analysis, feature engineering, signal processing, optimization methods, simulation methods, damage prognostics, data quality and imputation, and surrogate modeling. We can identify two ways to build analytics, either based on practical rules or a mathematical model.

Rule-based analytics

Rule-based analytics use knowledge about a variable or a particular feature to build a decision-based algorithm. Rule-based analytics can either use expert systems, classifiers, or rule-based ML. Rule-based analytics, for instance, can translate human knowledge or empirical rules into an algorithm.

In *Chapters 7* , *8* , *10* , and *12* , we already implemented a simple rule-based algorithm on AWS/Azure based on a simple threshold. A more complex algorithm might use fuzzy logic or Bayesian probabilities.

Model-based analytics

What is a model? A model is a mathematical and probabilistic relationship that exists between different variables.

Model-based analytics used with IIoT try to describe the relationship between input and output or the internal status of the monitored equipment. This relationship can either be based on a mathematical formula or inferred from data. In other words, we can distinguish between a physics-based model and a data-driven model.

Physics-based models

Physics-based models assume that you have a pretty detailed knowledge of the behavior of the equipment. Physics-based models join this knowledge about the equipment with the measured data to measure the performance of a piece of equipment and/or to predict its future behavior.

Compared to data-driven methods, physics-based analytics have several advantages:

- Physics-based methods require a relatively small dataset
- Physics-based methods can make a long-term prediction
- Physics-based models are normally more reliable

However, a disadvantage of this approach is that it requires in-depth knowledge of the problem to catch a physical failure.

Data-driven models

Data-driven models can use statistics, Artificial Intelligence (AI), standard ML, deep learning, generative AI, or general AI.

> ### *AIoT*

> ***Artificial Intelligence of Things*** *(* ***AIoT*** *) is the combination of AI technologies and IoT. AIoT aims to create more efficient IoT systems, enhance human-machine interactions, and improve data management and analytics.*

These techniques depend on collecting a history of failures, which requires volumes of data.

Without having a comprehensive understanding of the system, it can be hard to know how much data is good enough for a specific purpose. We suggest that you collect at least six months' worth of data with relevant events. In data-driven approaches, the most common technique is to use an artificial Neural Network (NN) (otherwise known simply as deep learning), in which a network model learns a way to produce a desired output. In the CBM, for instance, this might refer to the level of degradation of a turbine or the lifespan of a filter.

Supervised, unsupervised, and reinforcement learning

ML can be further divided into three classes: supervised, unsupervised, and reinforcement learning. In supervised learning , ML uses labeled datasets to train the model to predict the outcome. In unsupervised learning (such as clustering), models are fed with unlabeled data so that they can discover patterns without any explicit (human) guidance. In reinforcement learning , an agent learns how to perform a task through repeated *trial-and-error* interactions with a dynamic environment (typically a simulation or a real environment). Of course, it is possible to combine the techniques: generative AI uses semi-supervised learning and reinforces learning by human feedback. See *Chapter 14* for more details.

Hybrid model

A special case of a data-driven model combines the data-driven approach with physical constraints to combine a purely data-driven approach with a physics-based approach. The simpler way to combine the two methods is to generate synthetic data with a physical model to feed a data-driven model. Models based on Physics-Guided Neural Networks (PgNNs) introduce physical constraints on the loss function of NNs. Physics-Informed Neural Networks (Pi-NNs) are a class of ML models that have been developed to perform supervised learning tasks while respecting and incorporating known physical laws and principles.

These models leverage the power of NNs to learn complex non-linear relationships, while simultaneously ensuring that the learned solutions adhere to the underlying physical constraints expressed as general non-linear differential equations. Physics-Encoded Neural Networks (PeNNs) are advanced ML models that leverage specialized architectures to address challenges faced by traditional PNNs and PI-NNs.

Data-driven versus physics-based models

Theoretically, it is possible to identify model parameters when the amount of data entries is equal to the number of unknown model parameters. Unfortunately, due to noise in the data and the insensitivity of the context, a greater amount of data entries is required. In any case, the required amount of data samples in physics-based algorithms is fewer than that of data-driven methods.

The following decision table will help you choose between a physics-based, data-driven, or hybrid models:

High reliability Low reliability	
Physics-based analytics (simple rule-based analytics) A small amount of data	
Data-driven A large amount of data	

Table 13.1 – Decision table to choose between a physics-based, data-driven, or hybrid model

When the model is ready, we can test it with our test set to identify the accuracy of our model.

Another important feature we need to consider is the model's explainability; in other words, in some cases, we need to understand the cause of the prediction. Explainable AI (XAI) can help in this regard, but if we need full explainability, we should consider a physics-based model.

Building IIoT analytics

Identifying the right parameters and techniques is the most important part of building IIoT analytics. In 1977, John Tukey wrote the book *Exploratory Data Analysis* . The following method is based on Exploratory Data Analysis (EDA) and is applied to the new cloud DevOps requirements.

> ### *ML operability*

> *When we work with ML and AI we speak about* ***Machine Learning Operability*** *(* ***MLOps*** *). The word MLOps is a compound of ML and the continuous development practice of DevOps in the software field. MLOps aims to deploy and maintain ML models in production.*

The following are the steps that we need to accomplish:

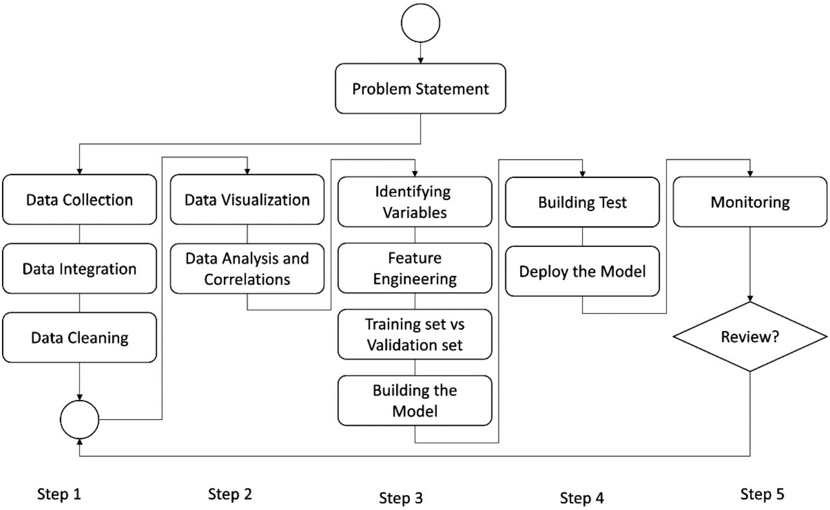

Problem Statement
Data Collection
Data Integration
Data Cleaning
Data Visualization
Data Analysis and Correlations
Identifying Variables
Feature Engineering
Training set vs Validation set
Building the Model
Building Test
Deploy the Model
Monitoring
Review?
Step 1
Step 2
Step 3
Step 4
Step 5

Figure 13.4 – Analytics development workflow

Step 0 – problem statement

Step 0, or the problem statement , always requires us to define the scope of the problem, the constraint of the problem, and the expected behavior. Then, we should align the business expectation with the technical expectation. It is a good practice to define the success of our analytics during this step. For example, you might accept 50% false positive results, or you might not want more than 2% false positive results.

Step 1 – dataset acquisition

During this step, we should acquire the largest amount of raw data and events. The following are three general intermediate steps:

1. Data collection : During this step, we collect data and related events or alerts.
2. Data integration (wrangling): During this step, we can convert the data format (using wrangling or munging) and integrate the data with the metadata or additional information.
3. Data cleaning : During this step, we need to prepare the data and remove spikes or bad data.

Step 2 – EDA

When the dataset is acquired, we need an investigatory phase. During this phase, we calculate the most relevant statistics of our data and plot this information on a trace plot, box plot, Pareto chart, multi-chart, or scatterplot. Finally, we can identify the correlation among variables.

Step 3 – building the model

When we have done the analysis of our data, we need to carry out the following steps:

1. Review the false positive rate.
2. Identify the variables that contribute to the expected outcome. This step is no longer mandatory thanks to deep learning, but in some cases, it can avoid overfitting in case of datasets being highly unbalanced or noisy:
    - Identify, reduce, and extract the features, a process known as feature engineering .
    - Feature extraction uses the raw dataset and builds calculated values (features) that are intended to be more informative and non-redundant.
    - Feature reduction reduces the number of features by selecting features or transforming them to a lower dimensional space without reducing the amount of information.

6. We then need to define the training set against the validation set. During this step, we allocate a portion of the dataset for the validation (holdout method) or repeat this process several times (cross-validation).
7. Finally, we can identify which technique to use to build our model. In this step, we should decide whether to use a physics-based or data-

driven model.

Step 4 – packaging and deploying (MLOps)

In this step, we package the analytics, defining the following:

- Unit test
- Regression test
- Performance test

We also need to wrap the data wrapper, which is the IN-OUT connector that transforms the input into the internal object structure and vice versa. Finally, we can deploy the algorithm.

Step 5 – monitoring

Don’t forget to periodically review the analytics to evaluate performance.

Understanding the role of the

infrastructure

In this book, in the *Developing our batch analytics with Airflow* section in Chapter 8 , as well as the *AWS IoT analytics* section in Chapter 10 , and the *Implementing a c loud industrial IoT solution with AWS* and *Performing a practical industrial IoT solution with Azure* sections in Chapter 12 , we discovered how to build analytics based on the cloud or, generally speaking, a centralized infrastructure. We also learned about cold and hot paths. From a theoretical standpoint, the implementation of analytics should be agnostic regarding where they are deployed and how we push data. Unfortunately, this is not always the case.

Analytics are strictly coupled with the support they want from the infrastructure. When analytics require a large amount of data to be sampled every millisecond, we might not be able to deploy on the cloud due to the restriction of the bandwidth. If the analytics that we are using require more data than produced in the last few seconds or minutes, a data stream platform may not be the right choice. If the analytics are stateless, we might be able to use serverless technologies. If a user needs to interact with these analytics, they cannot work easily in streaming mode.

In conclusion, the implementation of analytics depends strictly on how we want to consume data and how we want to interact with the other actors, device, and user.

Deploying analytics

Although analytics should be agnostic with regard to how the data is fed to the platform, we have to consider several potential pitfalls that can affect the efficiency of the analytics. There are several strategies that we can use to feed IIoT data to the platform:

- Bulk ingestion , for example, one file daily
- Small portions , for example, one file every five minutes
- Data streams , where files are fed continuously with a small latency

Data is also affected by several issues:

- It might be in the wrong order. For example, a data point at 18:00 might be sent at 18:10 and a data point at 17:59 might be sent at 18:11.
- It might be of a bad quality. It might have holes in it.
- It might have anomalous spikes in it.
- It might be frozen. This refers to a situation where you have a suspiciously flat number for a long time.

These issues are illustrated in the following diagram:

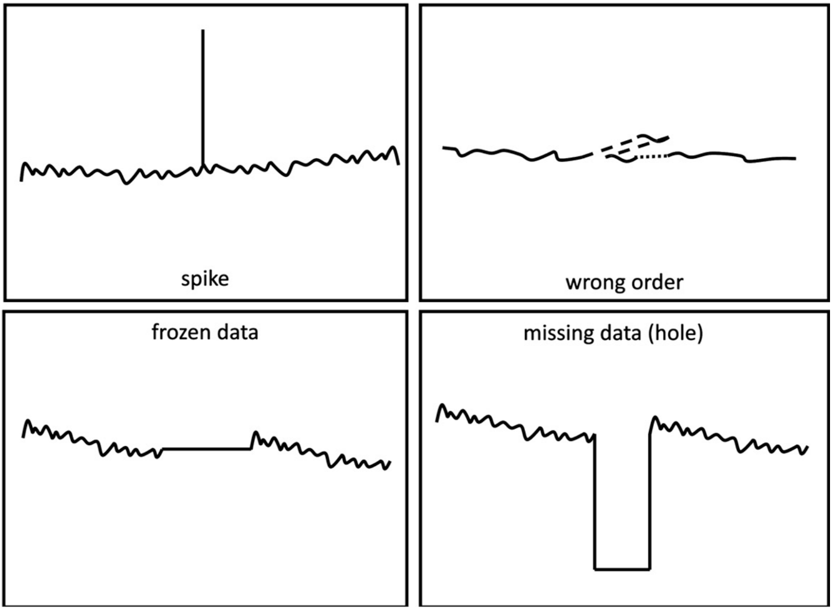


Figure 13.5 – Data being affected by issues

Data might also be delayed for a long period of time. We know this from personal experience in the oil and gas industry—one particular customer reactivated their connection to the cloud after six months of being disconnected and the data from the sensors filled the data lake in three days. Unfortunately, the analytics processed the data of the entire time period and detected a whole series of anomalies and alerts. These alerts were not useful at all because they were from the time in which the customer was disconnected, so the operations center was flooded with junk alerts.

To build a real IIoT platform, we have to develop our architecture with sufficient robustness to address these issues. For example, we can adopt a timeout for data that is too late, or we can pre-process and mark any data that is in the wrong order or frozen. Moreover, we can interpolate data before fueling the analytics to avoid holes.

In the previous section, we learned about the technologies required to build analytics. Now, we need to deploy them, assuming that the infrastructure supports our use case. We then need to define the method to trigger the analytics. We have three methods we can use to do this: stream analytics , micro-batch analytics , and condition-based analytics.

Streaming versus batch analytics

Streaming analytics are triggered when data flows into the system. This method is very good for simpler analytics, such as simple rules or signal processing. However, things become more complicated if analytics require the support of an additional data source or if they require data of a specific size.

An alternative to streaming is to schedule the analytics to run regularly after a set period of time, such as every 10 seconds. This involves checking the availability of the data and then pulling it from the data lake or the time-series database. We call these analytics micro-batch analytics.

Micro-batch analytics have a few great advantages:

- They are easy to debug and develop offline
- They can require a relatively large bunch of data
- They can re-process data when needed

Unfortunately, these analytics waste resources on checking the availability of data continuously. A good compromise is to use streaming analytics for simple rules and signal processing, as well as to also register the last timestamp of the data availability of the specific equipment.

The following diagram shows the proposed implementation:

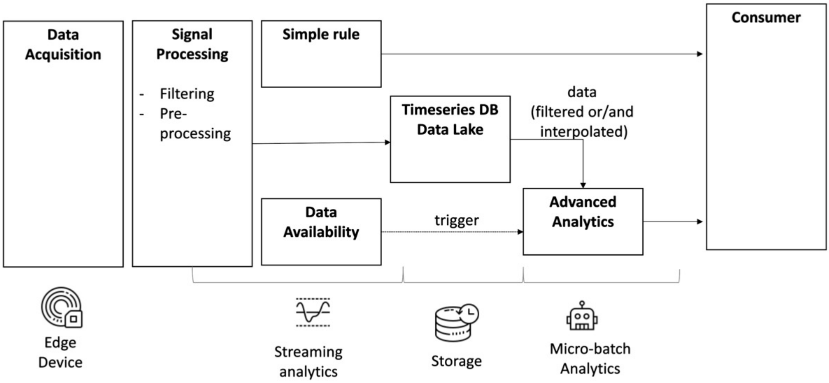

Data Acquisition
Signal Processing
- Filtering
- Pre-processing
Simple rule
Timeseries DB Data Lake
data (filtered or/and interpolated)
Data Availability
trigger
Advanced Analytics
Consumer
Edge Device
Streaming analytics
Storage
Micro-batch Analytics

Figure 13.6 – Batch and streaming analytics

Data is acquired from sensors and feeds the streaming analytics layer. Data points can be cleaned, and a simple rule can be applied. In parallel, we register the last timestamp, and we store the cleaned data in the storage layer. Micro-batch analytics are activated by the data's availability.

Condition-based analytics

Analytics can also be triggered by unpredictable events such as human interactions or rare events such as the abrupt shutdown of a piece of equipment that is being monitored. The previous framework remains valid; we simply trigger the execution based on a rule condition, which is known as condition-based analytics .

It's a different story, however, if a human wants to work with these analytics to perform, for instance, a what-if scenario, or to discover a specific pattern.

Interactive analytics

Interactive analytics represents a combination of queries, a user interface, and a core for the analytics. It is not possible to define the best method or technology to implement interactive analytics. The best suggestion is to keep the core of the analytics in a shared library and to develop the user interface with a small micro-app that uses a common framework such as Power BI, or Python-based libraries such as Dash, Streamlit, or Bokeh. Alternatively, we can use more advanced tools such as Jupyter (http://jupyter.org/).

Analytics on the cloud

Cloud analytics offer the best advantages, but also have two important disadvantages. They can be deployed and managed easily and they can scale rapidly. They can also work on a large amount of data over the entire fleet and cross-assets. However, they cannot process high-frequency data (more than 100 GB every hour with a sampling interval of less than 1 ms). This issue is not due to the cloud analytics themselves, but due to the bandwidth. The latency of the network causes a problem because cloud analytics cannot work with low latency. For these special cases, we prefer to run on the edge.

Analytics on the edge

On-edge analytics are the newest class of analytics in the IoT and the oldest class of analytics in the industrial sector. Indeed, industries started developing analytics on the controller from the beginning. Recently, the idea that everything should be cloud-based has become less important and vendors have started deploying cloud analytics on the edge. This is described in more detail later in this chapter.

There are three use cases in which it is important to deploy analytics on the edge:

- If you have high sampling data, such as the vibration of a turbine
- If you have a policy restriction regarding sending data outside the region
- If you require low latency

Let's discover the most interesting techniques to implement on-edge analytics.

Analytics on the controller

It is rare that cloud-based or edge-based analytics are allowed to perform actions on the controller in the industrial sector. On the contrary, we prefer that analytics that are optimized on the controller carry out operations that could impact production. This is due to issues of security and a lack of robustness. Moreover, these operations had already been made before the revolution of the IoT. Therefore, there are already widely tested technologies based on on-controller analytics .

Normally, these analytics work on a Real-Time Operating System (RTOS), which is very far from the high latency of the cloud. AWS, OEM's Cloud, and Google, however, are working to deliver an RTOS that works closely with an Edge device. Examples of this include AWS Greengrass and FreeRTOS.

Advanced analytics

Recently, advances in AI have caused the need for higher computational capabilities. Vendors such as NVIDIA and Apple have developed a special processor, Graphic Process Unit (GPU) and Neural

Process Unit (NPU), respectively, to support this new requirement. AWS, Azure, and GCP support this new processor on the cloud. Today, NVIDIA with its CUDA driver and HFX GPU is the leader in this sector. We will look more closely at these capabilities in *Chapter 15* .

OSA

The Open System Architecture (OSA) summarizes the concepts we discussed previously in a single picture. In the earlier *Prescriptive analytics* section, we highlighted how CBM is one of the most valuable aspects of IIoT analytics and how it can support insights from descriptive analytics all the way through to prescriptive analytics. The OSA for CBM is depicted as follows:

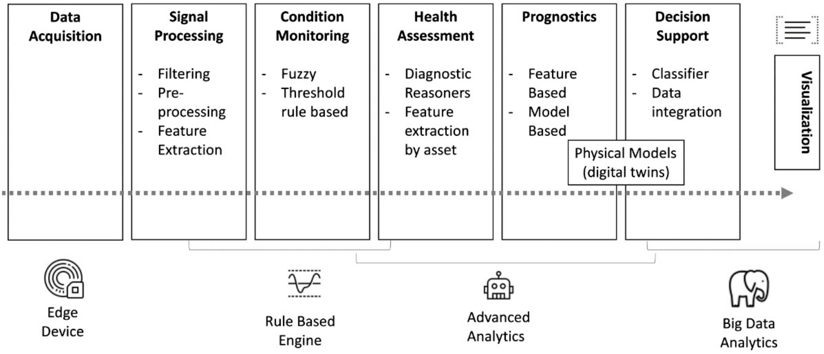

Data Acquisition
Signal Processing
- Filtering
- Pre-processing
- Feature Extraction
Condition Monitoring
- Fuzzy
- Threshold rule based
Health Assessment
- Diagnostic Reasoners
- Feature extraction by asset
Prognostics
- Feature Based
- Model Based
Decision Support
- Classifier
- Data integration
Physical Models (digital twins)
Visualization
Edge Device
Rule Based Engine
Advanced Analytics
Big Data Analytics

Figure 13.7 – The OSA model applied to the IIoT

The OSA for CBM is the most popular framework. It consists of six layers— data acquisition , signal processing , condition monitoring , health assessment , prognostics , and decision support . All these steps can be implemented in cloud-based architecture and partially on-premises, as explained in the previous sections. Signal processing and condition monitoring can be done using on-stream analytics or rule-based engines. The health assessment and prognostics phases, however, require advanced analytics based on ML or physics-based models. The last step can be either on-demand analytics or a big data query.

Analytics in practice

Let’s now put what we have learned into practice. We are going to build a diagnostic analytic and a predictive analytic. We will develop an anomaly detection algorithm for an airplane and a predictive algorithm for an oil and gas refinery. We want to remain as generic as possible, so we won’t make any assumptions about the system that we are going to monitor.

We will develop these two use cases with Python, SciPy, NumPy, Seaborn, and pandas. We will assume that Python 3.9+ is already installed on your system and that Jupyter is running on your PC (see the first paragraph in the *Jupyter* subsection earlier on how to start the notebook).

Anomaly detection in practice

For this exercise, we will use a free dataset for predictive maintenance (the dataset is based on

Azure Dataset and can be found at https://learn.microsoft.com/en-us/azure/architecture/industries/manufacturing/predictive-maintenance-overview).

Steps 0 and 1 – problem statement and the dataset

The dataset can be downloaded from, the official repository at https://ti.arc.nasa.gov/opensource/projects/. The challenge is to predict the anomalies.

The data is provided in a CSV format where the four columns are datetime, rotate, volt pressure of the machine, and the expected anomaly: 1 if anomalous and 0 if not anomalous.

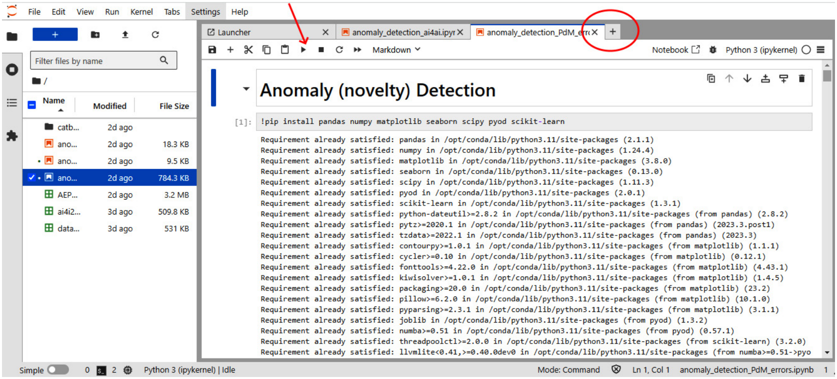
Problem statement

For our exercise, we want to calculate the anomalies in the time series to understand the potential issues that might occur during a standard operability.

Preparing the environment

First, we need to prepare our environment. From the command console, start Jupyter Notebook, as shown in the first paragraph of this chapter.

Download the dataset from the official repository into the current folder. Open the browser on **`http://localhost:8888/`** and start a new Python notebook (see *Figure 13 .8*). Then, to run a cycle cell, click on the play arrow.


File
Edit
View
Run
Kernel
Tabs
Settings
Help
Filter files by name
Name
Modified
File Size
catb...
2d ago
ano...
2d ago
18.3 KB
ano...
2d ago
9.5 KB
ano...
2d ago
784.3 KB
AEP...
2d ago
3.2 MB
ai4i2...
3d ago
509.8 KB
data...
3d ago
531 KB
Launcher
anomaly_detection_ai4ai.ipyr
anomaly_detection_PdM_errc
Markdown
Notebook
Python 3 (ipykernel)
Simple
Python 3 (ipykernel) | Idle
Mode: Command
Ln 1, Col 1
anomaly_detection_PdM_errors.ipynb

Anomaly (novelty) Detection

```
[1]: !pip install pandas numpy matplotlib seaborn scipy pyod scikit-learn
```

```
Requirement already satisfied: pandas in /opt/conda/lib/python3.11/site-packages (2.1.1)
Requirement already satisfied: numpy in /opt/conda/lib/python3.11/site-packages (1.24.4)
Requirement already satisfied: matplotlib in /opt/conda/lib/python3.11/site-packages (3.8.0)
Requirement already satisfied: seaborn in /opt/conda/lib/python3.11/site-packages (0.13.0)
Requirement already satisfied: scipy in /opt/conda/lib/python3.11/site-packages (1.11.3)
Requirement already satisfied: pyod in /opt/conda/lib/python3.11/site-packages (2.0.1)
Requirement already satisfied: scikit-learn in /opt/conda/lib/python3.11/site-packages (1.3.1)
Requirement already satisfied: python-dateutil>=2.8.2 in /opt/conda/lib/python3.11/site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /opt/conda/lib/python3.11/site-packages (from pandas) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in /opt/conda/lib/python3.11/site-packages (from pandas) (2023.3)
Requirement already satisfied: contourpy>=1.0.1 in /opt/conda/lib/python3.11/site-packages (from matplotlib) (1.1.1)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.11/site-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /opt/conda/lib/python3.11/site-packages (from matplotlib) (4.43.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /opt/conda/lib/python3.11/site-packages (from matplotlib) (1.4.5)
Requirement already satisfied: packaging>=20.0 in /opt/conda/lib/python3.11/site-packages (from matplotlib) (23.2)
Requirement already satisfied: pillow>=6.2.0 in /opt/conda/lib/python3.11/site-packages (from matplotlib) (10.1.0)
Requirement already satisfied: pyparsing>=2.3.1 in /opt/conda/lib/python3.11/site-packages (from matplotlib) (3.1.1)
Requirement already satisfied: joblib in /opt/conda/lib/python3.11/site-packages (from pyod) (1.3.2)
Requirement already satisfied: numba>=0.51 in /opt/conda/lib/python3.11/site-packages (from pyod) (0.57.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /opt/conda/lib/python3.11/site-packages (from scikit-learn) (3.2.0)
Requirement already satisfied: llvmlite<0.41,>=0.40.0dev0 in /opt/conda/lib/python3.11/site-packages (from numba>=0.51->pyo
```

Figure 13.8 – Open a new Jupyter Notebook and run the cell

Open a new cell and install the required packages:

```
%pip install pandas numpy matplotlib
scikit-learn catboost
```

It will take a while to complete but after the first installation, you can avoid having to repeat this step.

Let's start writing the following lines of code to import the dataset:

```
import pandas as pd
df =
pd.read_csv('data_PdM_errors.csv')
df['datetime'] =
pd.to_datetime(df['datetime'])
```

Then run the cell from Jupyter Notebook (see Figure 13 .8).

Step 2 – EDA

The first step is to show the data:

```
import matplotlib.pyplot as plt
# showing dataset
df.plot(x='datetime', kind='line',
subplots=True)
plt.show()
```

The result will be similar to the following:

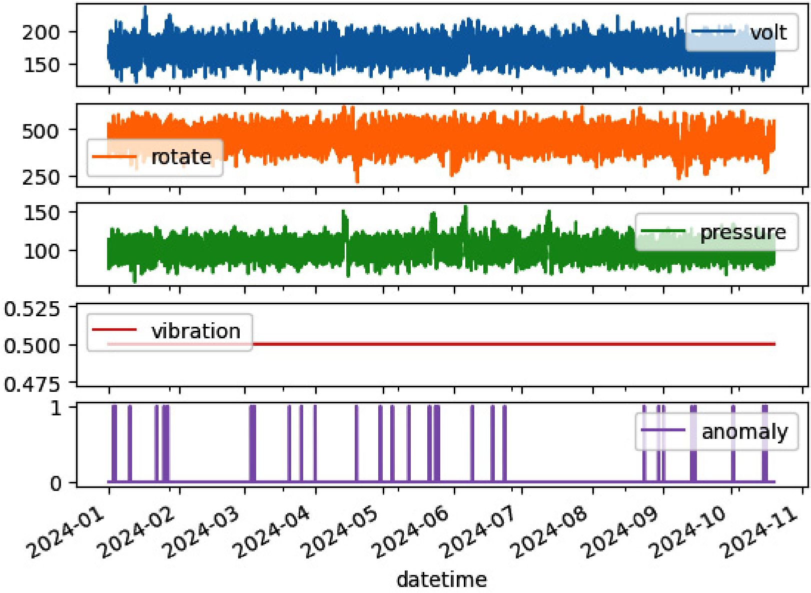


Figure 13.9 – Time series of our machine; the last time series shows the anomalous data

The dataset is good, so we don't need to clean it. We can now analyze the standard deviation of a few variables. The following code performs this analysis:

```
# analysis of variance
df_std = df.std()
print(df_std)
```

All three variables show a high standard deviation except for vibration; indeed, vibration is a constant so it cannot provide any kind of information. We can, therefore, remove this variable:

```
#removing un-usefull vars
df=df.drop(['vibration'], axis=1)
```

We can now analyze the correlation between the sensors. The following code calculates the Pearson correlation. The Pearson correlation is a measure of the linear correlation between two variables:

```
from seaborn import heatmap
heatmap(df.corr(),annot=True,cmap='Blues',
fmt='.2f')
```

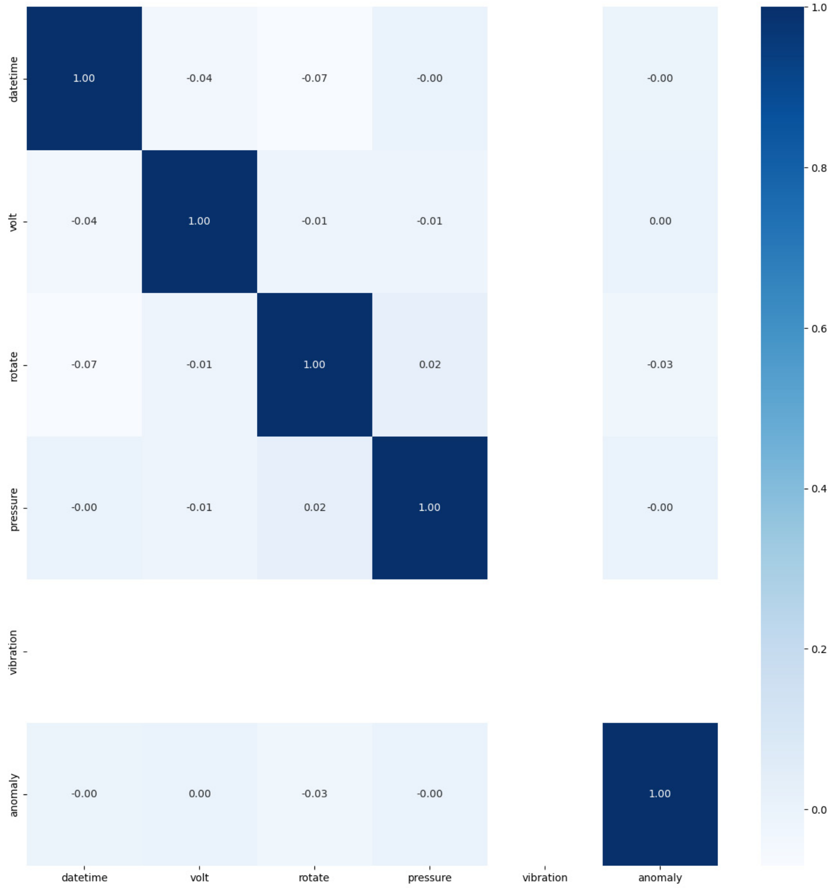


Figure 13.10 – Correlation matrix

The correlations found between the variables suggest that we need to carry out further investigation because no strong correlation has been found among variables and between anomalies and variables. Therefore, except for *vibration* (which is not useful), all the variables will contribute to identifying the anomalies.

Now we can split our dataset into two parts: input (**`X`**) and expected output (**`y`**). We will use **`y`** only to test the output and not to build the model:

```
# separate into input and output
variables
X = df[['volt', 'rotate',
'pressure']].values
y = df[['anomaly']].values
```

Finally, we can split between **`train`** and **`test`** . We will use the **`train`** set to build the model and the **`test`** set only to validate the prediction of the model:

```
idx_split=6500
X_train, y_train = X[0:idx_split,:],
y[0:idx_split]
X_test, y_test = X[idx_split:,:],
y[idx_split:]
```

Typically, we divide the data into training and testing sets, allocating 70% for training and 30% for testing. When we work with time series, however, we need to keep the time dependency among the samples, so we kept the last 500 samples just for **`test`**.

Step 3 – building the model

Let's build our algorithm by using the previous steps.

We decided to use an unsupervised method based on a moving average filter.

Building the algorithm

In this simplified exercise, we will use a statistical method that's been applied to a moving average algorithm. The algorithm evaluates the standard deviation (**`std`**) in a window of 60 points and checks the data points that exceed **`2*std`**:

```
import numpy as np
def moving_average(data, window_size):
    window =
np.ones(int(window_size))/
float(window_size)
```

```
    return np.convolve(data, window,
'same')
def search_anomalies(x, window_size,
sigma=1.0):
    avg = moving_average(x,
window_size).tolist()
    residual = x - avg
    # Calculate the variation in the
distribution of the residual
    std = np.std(residual)
    anomalies=[]
    i=int(window_size)
    for y_i, avg_i in zip(x, avg):
        if (y_i > avg_i + (sigma*std))
| (y_i < avg_i - (sigma*std)):
            anomalies.append(1)
        else:
            anomalies.append(0)
    return np.array(anomalies)
```

This algorithm has been applied to the `tw` variables, **`volt`** and **`pressure`**:

```
volt = df['volt'].values
pressure = df['pressure'].values
anomalies_volt =
search_anomalies(volt, window_size=60,
sigma=2)
anomalies_pressure =
search_anomalies(pressure,
window_size=60, sigma=2)
```

Finally, we can write a simple function to show the accuracy of our model and see the anomalies identified by our model versus the real one:

```
from sklearn.metrics import
confusion_matrix, accuracy_score
import matplotlib.pyplot as plt
def show_anomalies(X_test, y_test,
y_predicted, anomalous=1, title=''):
    t = np.array([i for i in
range(len(y_test))])
    idx_predicted =
y_predicted==anomalous
    idx_real = y_test==anomalous
    plt.figure(figsize=(10,5))
    plt.xlabel('samples')
    plt.ylabel(title)
    plt.plot(t,X_test,'tab:gray')
    plt.plot(t[idx_predicted],
X_test[idx_predicted],'rD',
label="predicted anomaly")
    plt.plot(t[idx_real],X_test[idx_real],'
label="real anomaly")
    plt.legend()
    plt.show()
    print('accuracy:',accuracy_score(idx_re
    print('confusion matrix:
\n',confusion_matrix(idx_real,idx_predicted
```

Let us see the anomalous samples of the volt variable:

```
show_anomalies(volt[idx_split:],
y[idx_split:],
anomalies_volt[idx_split:],
title='volt (V)')
```

The result of the previous code is shown in the following graph:

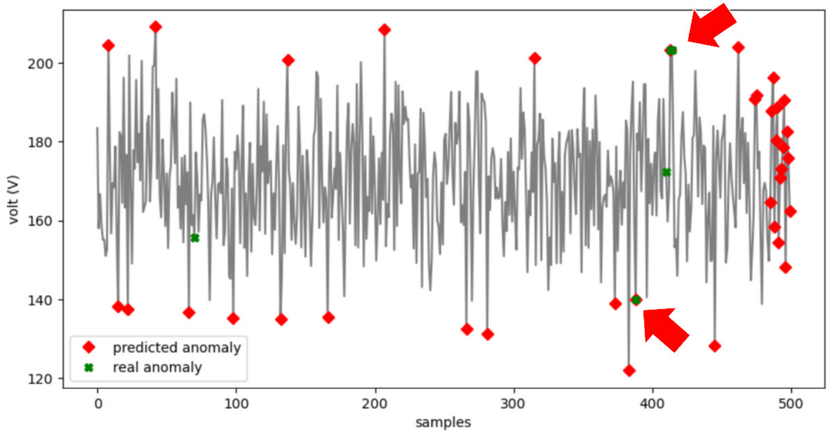


Figure 13.11 – Two anomalies identified by the algorithm

Excluding the start and the end of the time series, we can accept only two anomalies. Indeed, despite the accuracy being very high (92.6%), there are a lot of false positives and three false negatives. This result is highlighted by the confusion matrix:

	Not anomalous samples predicted by the model	Anomalous samples predicted by the model
Not anomalous samples	461	3 (false positive)
Anomalous samples	2 (false negative)	3

Table 13.2 – Confusion matrix

Despite false positives not being a big issue, false negatives can cause serious damage. Therefore, anomaly detection algorithms prefer to pay attention and not miss anomalies; these behaviors can occur on a greater number of potential anomalies. In defense of our algorithm, we must consider that our model knows nothing about the physics of the machine, nor does it know anything about the previous anomalies, but simply observes what is “abnormal.” In the next sections, we will try to use a more complex model that makes use of unsupervised ML.

Step 4 – packaging and deploying

We can now develop a unit test and a regression test to check the quality of our analytics. The following algorithm develops a simple unit test that checks the anomaly detection procedure that we have developed with a predefined dataset. The algorithm just checks whether the two samples, **`30`** and **`20`**, are identified as anomalous:

```
import numpy as np
def test_search_anomalies():
    data=np.array([10, 10, 10, 10, 30,
20, 10, 10])
    predicted=search_anomalies(data,
2)
    assert predicted[1]==0
    assert predicted[4]==1
    assert predicted[5]==1
```

It is important to build a small dataset where we know the expected outcome to avoid potential performance loss due to minimal change. In our experience, the most common errors and bugs are never made in the model but in the data acquisition, conversion, and cleaning phases.

To deploy the analytics on the cloud, we have to decide whether to use a cold path or a hot path. The developed algorithm doesn't need a lot of data and should run on the data that we have available. We can use a stream-based infrastructure and deploy the algorithm directly onto the data stream.

Step 5 – monitoring

It is a good idea to review the performance of the algorithm quarterly, calculating the false positives and negatives with the newly acquired dataset.

Anomaly detection with unsupervised ML

We now want to apply a more complex algorithm to our previous algorithm. We will keep the same assumptions as we had before and only implement the new algorithm. Steps 0-2 are the same as those described in the *Anomaly detection in practice* section.

Step 3 – building the model with ARIMA and One-Class Support Vector Machine

As mentioned previously, one of the problems of the proposed algorithm is the lack of information about the dataset—we do not know the normal operation of the airplane. For this reason, we have implemented a simple method that identifies when a data point falls outside the standard deviation.

We can improve our algorithm using an unsupervised ML algorithm. A particularly interesting algorithm is the One-Class Support Vector Machine (OCSVM). The OCSVM splits data points into two different classes— *good* or *bad* . It requires a training dataset to build the boundaries. We can use the full dataset for the training phase and then evaluate each point to see whether it falls into the good class or not. Our dataset has to be normalized because it reports a reasonability trend. To do this, we can build an Autoregressive Integrated Moving Average (ARIMA) model and then analyze the residuals. This trick is a common way to remove trends or seasonality from a dataset.

The following code defines a function to extract residuals from the dataset and to apply the OCSVM:

```
from statsmodels.tsa.arima.model
import ARIMA
from matplotlib import pyplot
import numpy as np
from sklearn.svm import OneClassSVM
def ARIMA_residuals(series):
    # fit model
    model = ARIMA(series,
order=(5,1,3))
    model_fit = model.fit()
    print(model_fit.summary())
    # plot residual errors
    residuals =
pd.DataFrame(model_fit.resid)
    return residuals.T
def search_anomalies_OCSVM(X_train,
X_test):
    clf = OneClassSVM(kernel="rbf",
tol=0.0001, nu=0.1).fit(X_train)
    y_pred = clf.predict(X_test)
    anomalies=[ 0 if _x > 0 else 1 for
_x in y_pred]
    return np.array(anomalies)
```

Now, we need only to bring these two functions together in a single cell:

```
X_residual=np.vstack((
ARIMA_residuals(volt),
             ARIMA_residuals(rotate),
             ARIMA_residuals(pressure))).T
anomalies=search_anomalies_OCSVM(X_residual
X_residual)
```

The first line of the preceding code computes ARIMA's residuals and puts them in an array. We then apply the OCSVM clustering algorithm to identify potential anomalies.

We increased the number of anomalies detected from two to three with an accuracy of 90%.

Not anomalous samples predicted by the model	Anomalous samples predicted by the model
450	Not anomalous samples (false positive)

2Anomalous samples (false negative)			

Table 13.3 – Confusion matrix

The basic idea is to remove the global trend of the data points, convert the dataset into a cluster of points around the 0-axis, and then identify the anomalies using a clustering algorithm such as OCSVM. The following graph shows the scatter plot of ARIMA's residuals and the outliers identified by OCSVM:

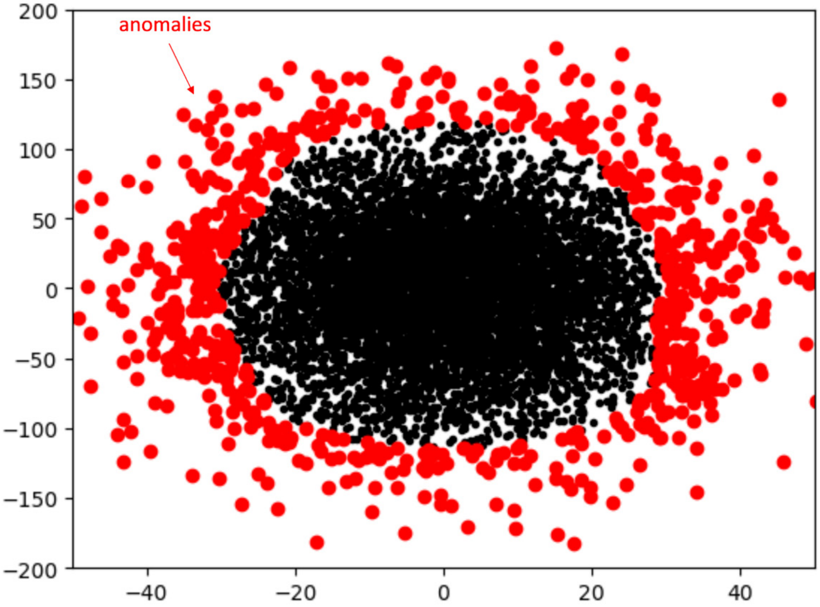


Figure 13.12 – Scatter plot of ARIMA's residuals and outliers

In this exercise, we have used very simple algorithms to explain the main concepts of anomaly detection. Classic ML models, such as Isolation Forest, are more robust. Finally, deep NNs, Variational Autoencoder (VAE), and diffusion models can identify complex anomalous behavior.

Step 3 – building the model with Isolation Forest

In the previous exercise, we used ARIMA to perform feature engineering and to calculate residuals to feed a second algorithm. Isolation Forest is a more powerful algorithm based on trees.

To build an anomaly detection based on Isolation forest, we first need to train our model on the `train` set. The following code uses the official package of scikit-learn to train the model. On Jupyter Notebook, add the following lines of code and run the cell:

```
from sklearn.ensemble import
IsolationForest
clf =
IsolationForest(contamination=float(0.05))
clf.fit(X_train)
```

Then, use the model to predict the outcome on the **`test`** set. On Jupyter Notebook, add the following lines of code and run the cell:

```
anomalies = clf.predict(X_test)
anomalies = np.array([1 if _x <
0  else 0 for _x in anomalies])
```

Amazing job! We got 96% accuracy, and we detected four anomalies versus five.

	Not anomalous samples predicted by the model	Anomalous samples predicted by the model
Not anomalous samples	481	1 (false positive)
Anomalous samples	1 (false negative)	4

Table 13.4 – Confusion matrix

Finally, show the anomalies detected:

```
show_anomalies(X_test[:,0], y_test,
anomalies)
```

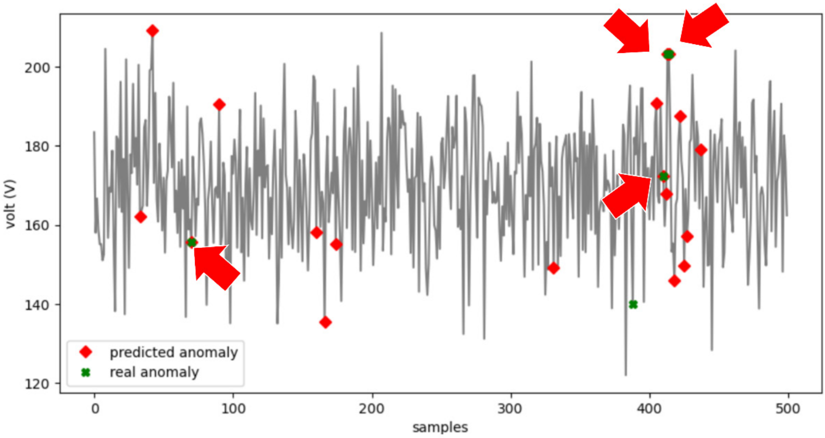


Figure 13.13 – Anomalies detected by Isolation Forest

This exercise taught us that in statistical and physical models, we need to do a good job of feature engineering, while in ML, we let the algorithm do it for us. The only critical operation is to tune the parameters.

Consideration about hyper-parameter tuning

In this exercise, we choose default parameters for our ML model. However, normally, we keep a small subset from the training set to fix the hyper-parameters of the model.

Anomaly detection with supervised ML

In the previous exercise, we used an unsupervised approach in which we knew nothing about the anomaly to look for; in this case, the anomaly is defined as “non-normal behavior.” This approach is also called novelty detection . However, when we have a good sample of already classified anomalies, we can reuse them to train an ML algorithm.

Steps 0 and 1 – problem statement and dataset

In this case, we will use a dataset with more multiple anomalies. In the dataset, we can find five inputs and five types of anomalies:

- Input variables consist of numerical value and label value. They are as follows: product ID , type , air temperature [K] , rotational speed [rpm] , and tool wear [min] .

- When it comes to output variables, the anomalies to predict are encoded in five columns ranging from `0` (not anomalous) to `1` (anomalous). They are Tool Wear Failure (TWF), Heat Dissipation Failure (HDF), Power Failure (PWF), Overstrain Failure (OSF), and Random Failures (RNF).

The dataset was published at the AI4Ai conference and is available on this book's official repository: S. Matzka, *Explainable Artificial Intelligence for Predictive Maintenance Applications* , 2020 Third International Conference on Artificial Intelligence for Industries (AI4I), 2020, pp. 69-74, **`https://ieeexplore.ieee.org/document/9253083`** .

To start working with the dataset, open a new Jupyter notebook and add a new cell with the following code:

```
import pandas as pd
df = pd.read_csv('ai4i2020.csv')
```

Then define the input variables (**`X`**) and the output variables (**`y`**). In the cell you just created, add the following line and execute the cell:

```
cat_features=['Product ID', 'Type']
X_features= cat_features +  ['Air
temperature [K]',
       'Process temperature [K]',
'Rotational speed [rpm]', 'Torque
[Nm]',
       'Tool wear [min]']
y_features=[ 'TWF', 'HDF', 'PWF',
'OSF', 'RNF']
X=df[X_features]
y=df[y_features]
```

Step 2 – EDA

Before starting to build our model, we need to understand our target variables. Add a new cell on the notebook and write and execute the following code:

```
y.value_counts()
```

The output should be similar to the following:

```
TWF  HDF  PWF  OSF  RNF
0    0    0    0    0      9652
     1    0    0    0       106
     0    1    0    0        80
          0    1    0        78
1    0    0    0    0        42
0    0    0    0    1        18
          1    1    0        11
     1    0    1    0         6
          1    0    0         3
1    0    0    1    0         2
               0    1         1
          1    1    0         1
```

As you can see, the majority of the data is not anomalous. This one is a typical situation in anomaly detection algorithms; in fact, there are very few anomalies. Fortunately, compared to the previous case (where the anomalies were only 0.4% of the total), in this case, we have a more significant sample, and the anomalies are around 3.4%. Therefore, we can attempt a supervised approach.

Note that a simple rule that always predicts *not anomalous* would achieve 96.6% accuracy, so it is necessary to ask the ML algorithm to pay more attention (give more weight) to anomalies.

So, now we split **`train`** and **`test`** , and we will use the **`train`** set (80% of the original dataset) to train the algorithm and the **`test`** set (20% of the original dataset) to validate the prediction:

```
from sklearn.model_selection import
train_test_split
X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.2,
random_state=0)
```

Step 3 – building the model

Now we can build our model. We will build a model for every type of anomaly. From Jupyter Notebook, open a new cell and add the following code. This code will import the required packages:

```
from catboost import
CatBoostClassifier
from sklearn import metrics
import numpy as np
```

Now define the model; we are using **`CatBoostClassifier`** because it is very easy to configure. Indeed, an ML model can work with categorical data (string) and we need to convert it to numeric; **`CatBoost`** does the encoding automatically. It only requires the name of the categorical variable. We have already defined the categorical **`cat_features`** variables in the previous section (*Step 1*); now, we can use the variable.

In the cell you just created, add the following line and execute the cell:

```
# set the current type of anomaly
on="HDF"
_y_train =
y_train[on].values.flatten()
clf = CatBoostClassifier(
        iterations=300,
        loss_function='Logloss',
        cat_features=cat_features,
        auto_class_weights='Balanced').fit(
_y_train)
```

Finally, evaluate our model. Add a new cell add the following line and execute the cell:

```
_y_val = y_val[on].values.flatten()
y_pred = clf.predict(X_val)
print(metrics.accuracy_score(_y_val,
y_pred))
print(metrics.balanced_accuracy_score(_y_va
y_pred))
print(metrics.confusion_matrix(_y_val,
y_pred))
```

The output should be similar to the following:

```
0.995
0.9974
[[1968   10]
 [   0   22]]
```

Fantastic! With just a few clicks, we found all 22 anomalies, caught 11 false anomalies, and achieved an accuracy (standard and balanced) of around 99%.

Repeat the exercise for all four other types of anomalies by changing the **`on`** variable: the values to use are **`'TWF'`**, **`'HDF'`**, **`'PWF'`**, **`'OSF'`**, and **`'RNF'`**.

More or less, the results will be compatible except for the RNF anomaly. In this case, we will not achieve more than 56% accuracy. This confirms what we discovered before: if the anomaly is known, the supervised approach is the best, but when the anomalies are rare or unknown, we must proceed with an unsupervised approach as we did in the previous exercise.

> ### *Metrics*

> *In classification algorithms, accuracy is an insufficient metric. In fact, in cases of strong data imbalance, it could be misleading. In this case, we used the balanced accuracy, which is the average of the accuracies between the two classes. However, there are more appropriate metrics such as precision, recall, and AUC. For a slightly more comprehensive list, see the scikit-learn metrics (https://scikit-learn.org/stable/api/sklearn.metrics.html).*

Explaining the anomaly

We have detected the anomaly, but the cause remains unclear. We can use a `CatBoost` feature that allows us to study the most important measure contributing to these anomalies. The following lines of code allow us to highlight the main contribution to the anomalies. Add the following three lines of code to explore our anomaly. From Jupyter, add a new cell add the following line and execute the cell:

```
feature_importance =
clf.feature_importances_
sorted_idx =
np.argsort(-1*feature_importance)
for col,imp in
zip(X_train.columns[sorted_idx],
feature_importance[sorted_idx]):
    print(f"{col} is {int(imp)}%
important on {on}"))
```

The code will sort the contributors of the anomaly from most to least important. The expected output is as follows:

```
Rotational speed [rpm] is 37% important
on HDF
Air temperature [K] is 33% important on
HDF
Process temperature [K] is 16% important
on HDF
Torque [Nm] is 6% important on HDF
Tool wear [min] is 2% important on HDF
Type is 2% important on HDF
Product ID is 0% important on HDF]
```

Indeed, the HDF anomaly is due to heat dissipation.

Therefore, it makes sense that the rotation speed and the two temperatures are the main contributors to the anomaly.

This class of analysis is called XAI. There are various techniques to apply XAI Shapely, Permutation Importance, and LIME (see also *Interpretable Machine Learning with Python, Second Edition* by Serg Masís).

More about anomaly detection

We have concluded our anomaly detection exercises. First, we developed an algorithm based on a physical concept. Then, we used a statistical approach based on ARIMA, and OCSVM then we used two unsupervised and supervised ML algorithms.

We learned that when we have a good sample of anomalies, we can use a supervised approach based on the old anomalies, but in general, this situation is very rare, and we have to use unsupervised or semi-supervised algorithms. A good list of advanced anomaly detection algorithms is offered by the pyOD Python package (https://pyod.readthedocs.io/).

In the next section, we will analyze how to predict future scenarios in order to optimize our production.

Predictive production analytics in practice

In the previous exercise, we used ARIMA to build a model of the signal. ARIMA (and its variants, AR, MA, ARIMAX, and so on) is one of the most common statistical algorithms to work with time series. In this exercise, we would like to use a more advanced tool called Prophet , developed by Meta. Prophet (https://facebook.github.io/prophet/) is a very powerful library, outperforming ARIMA, and most of the best predictive algorithms.

Steps 0 and 1 – problem statement and dataset

The dataset for this problem can be downloaded by forking the official book repository at https://github.com/PacktPublishing/Hands-On-Industrial-Internet-of-Things-Second-Edition . The dataset contains the power consumed in the USA.

Our problem statement is to predict the future (max 24 hours) demand of the USA, given the information about the last months, so that we can adjust our power plant accordingly to expected power consumption.

Download data from https://www.kaggle.com/datasets/robikscube/hourly-energy-consumption .

We can reuse the previous code and change the previous code as follows:

```
import pandas as pd
df = pd.read_csv('data_AEP.csv')
df = df.rename(columns={'Datetime':
'ds', 'AEP_MW': 'y'})
```

```
df = df[0:3000]
```

Step 2 – EDA

The data consists of a single variable varying from 0 to 3604 MW; we can explore the details of our variable:

```
df.describe()
```

Step 3 – building the model

We can now implement our model with Prophet:

```
from prophet import Prophet
period = int(24) # hours
m = Prophet()
m.fit(df[:-1*period])
```

Now we can see the outcome:

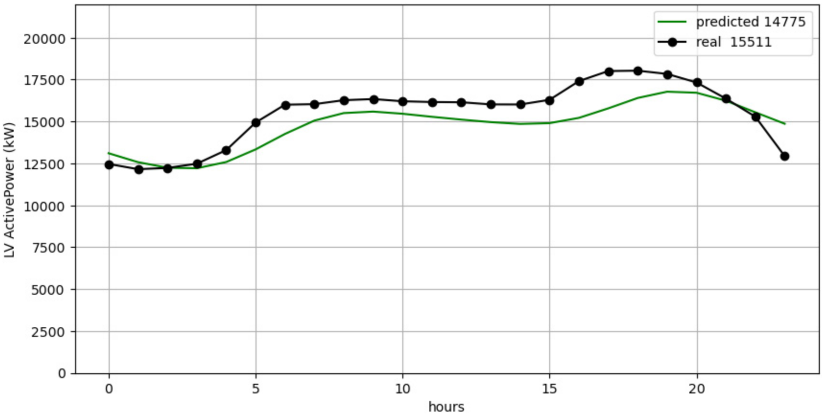


Figure 13.14 – Predicting electricity demand with Prophet: R2 score is 0.59

Steps 4 and 5 – packaging, deploying, and monitoring

For the last steps, follow the procedure that we took during the *Anomaly detection in practice* subsection.

We used Prophet to predict future a scenario based only on the trend of the demand. In the next section, we will analyze how to explain anomalies using language models.

Prescription using language models

ML algorithms are frequently employed in prescriptive analytics to analyze vast datasets more quickly and efficiently than humans can. By utilizing a “what-if” statement, these algorithms look at data to build insights on specific combinations of criteria. In this exercise, we will use generative AI and in particular, a Large Language Model (LLM) to analyze a list of failure events based on the previous dataset already analyzed in the *Anomaly detection with supervised machine learning* section.

From an implementation point of view, we will implement a Retrieval Augmented Generation (RAG) system. A RAG system indexes all documents, searches for the most similar content to the given question, then uses this content as a new context for the LLM to generate the response.

Steps 0 and 1 – problem statement and dataset

The dataset is composed of 348 events written in natural language in the following format:

```
"Found <type of anomaly> anomaly for
Product <code>:  <description>. The
cause is <measure> which is <value>. A
second cause can be <measure> which is
<value>"
```

We want to reuse this knowledge base to ask for a potential failure in some circumstances, for example, when **`Rotational speed`** is about a given value.

Step 3 – building the model

The first step is to install the **`llama_index`** and **`transformers`** packages. From Jupyter, open a new cell and install the following packages. In the cell you just created, add the following line and execute the cell:

```
!pip install llama_index llama-index-
llms-huggingface llama-index-
embeddings-huggingface accelerate
```

```
bitsandbytes
```

Now we can import the required packages. Create a new cell, add the following lines, and execute the cell:

```
from llama_index.core import
PromptTemplate
from llama_index.llms.huggingface
import HuggingFaceLLM
from llama_index.core import Settings
from llama_index.core.prompts.prompts
import SimpleInputPrompt
```

Now read the documents. Add a new cell, write the following line of code, and execute the cell:

```
from llama_index.core import
SimpleDirectoryReader
```

Now read the documents. Add another cell, write the following lines of code, and execute the cell:

```
reader = SimpleDirectoryReader(
    input_files=["events.txt"],
    required_exts=[".txt"],
    num_files_limit=100, # Maximum
number of files to load
    encoding="latin-1",
)
documents = reader.load_data()
```

Now we need to set up the LLM. For the sake of simplicity, we will use ChatGPT. Connect to https://platform.openai.com/api-keys and generate a new API key, then set the new API key to the **`environment`** variable. In a new cell, write the following line of code and execute the cell:

```
%env OPENAI_API_KEY=<your key>
```

> ### *Use local LLM*

> *If you don't want to use ChatGPT, you can use LLAMA-3 or Mistral. In the official repository of this book, you can find a simple implementation using Microsoft Phi-3 mini.*

Load the index directly from **`Documents`** . Add a new cell, write the following lines of code, and execute the cell:

```
from llama_index.core import
VectorStoreIndex
```

```
index =
VectorStoreIndex.from_documents(documents,
show_progress=True)
```

Finally, prepare your question. In the previous cell, write the following lines of code to get your insights:

```
query_engine = index.as_query_engine()
response = query_engine.query("When
Rotational speed is about 1342 what
can be the anomaly?")
print(f"{response}")
```

The expected output will be as follows:

Based on the provided context, when the rotational speed is about 1342 rpm, the anomaly that can be observed is a heat dissipation failure (HDF). This anomaly occurs when heat dissipation causes a process failure, if the difference between air- and process temperature.

In this final exercise, we saw how to use RAG to extract insights from a time series of past events. Of course, it is possible to extend this system to machine manuals, field reports, and other tickets. If you want to know more about RAG, see *Building Data-Driven Applications with LlamaIndex* by Andrei Gheorghiu.

Summary

In this chapter, we explored the most important classes of analytics in IIoT from a theoretical point of view. We looked at the most important use cases, as well as different classes of analytics and the importance of CBM.

We also discussed the relationship of analytics with data in terms of model accuracy and data processing; we also introduced the new concept of AIoT. Finally, we implemented a diagnostic algorithm and a predictive analytic model. We learned that anomaly detection and production prediction are two of the most common IIoT algorithms. Finally, we implemented a prescriptive analytic based on RAG to analyze old events.

In the next chapter, we will focus on the kernel of analytics—the digital twin.

14

Implementing a Digital Twin – Advanced Analytics

In the previous chapter, we investigated some of the most interesting classes and use cases to do with IIoT analytics. We discovered that an analytic can be descriptive, diagnostic, predictive, or prognostic. We also mentioned the remaining useful life (RUL) of an asset or part within IIoT.

In this chapter, we will improve our knowledge of IIoT analytics, discover the concept of digital twins , and use more advanced technologies based on machine learning (ML) and deep learning (DL).

In this section, we will explore the following:

- Digital twins

- Artificial intelligence (AI), ML, and DL
- Practical examples with ML algorithms and a physics-based algorithm
- Platforms on which to build digital twins

Technical requirements

In this chapter, we need the following prerequisites:

- Docker : https://www.docker.com/
- Git : https://git-scm.com/downloads
- Python 3.9+ : https://www.python.org/downloads/

The code for this chapter is available in the official repository at https://github.com/PacktPublishing/Hands-On-Industrial-Internet-of-Things-Second-Edition .

Digital twins

Digital twins are broadly defined as virtual representations of physical assets, providing an accurate and detailed replica of their real-life counterparts to enable advanced analysis, simulation, and optimization. The first experiments with digital twins were developed by NASA to mirror outer space as a virtual image (Allen, B. Danette. *Digital twins and living models at NASA. Digital Twin Summit*. 2021). Later, digital twin technology was used for product design and manufacturing in techniques such as 3D prototyping.

As of now, there is no precise definition of the concept of a digital twin . The various existing definitions are defined by specific characteristics that derive from multiple use cases that employ the concept of digital twins.

When more complex objects began to produce data, digital twins moved beyond manufacturing into the domains of IIoT, AI, and data analytics. In this book, we accept the definition by Glaessgen and Stargel with a slight change (see *Digital Twin in Practice*):

> *A Digital Twin is an integrated multiphysics, multiscale, probabilistic simulation of an as-built vehicle or system that uses the best available physical models, sensor updates, fleet history, etc., to mirror the life of its corresponding flying twin. The Digital Twin is ultra-realistic [...] integrates sensor data [...] maintenance history and all available historical and fleet data obtained (Glaessgen, E., and Stargel, D. 2012. The Digital Twin Paradigm for Future NASA and U.S. Air Force Vehicles , in Structures, Structural Dynamics, and Materials and Co-located Conferences , American Institute of Aeronautics and Astronautics).*

Having a digital copy of a physical object gives data scientists the ability to optimize efficiency and create other what-if scenarios.

Digital twins can be used in the following sectors:

- Renewable energy : To emulate wind turbine plants and optimize production

- Manufacturing process : For designing, diagnostic purposes, degradation analysis, or what-if scenarios
- Oil, gas, and power generation : To optimize production
- Chemistry : To improve efficiency
- Aviation : For regulatory compliance management
- Automotive : For designing and testing different materials
- Construction : For designing or checking security events
- Healthcare : For diagnostic or educational purposes.
- Smart cities : For what-if scenarios, traffic flow analysis, energy efficiency, and emergency response
- Athletics : To simulate athletes' response

Now that we have understood what a digital model is for, in the following section, we will study the five main classes of digital twins.

Digital twins in practice

The development of a complete digital model can leverage many technologies (AI, physical models, 3D models, mathematical models, or chemical models). Therefore, developing a true Omni digital twin model can be complex. From the perspective of developing a minimum valuable solution or product, you can decide to limit the scope to parts, to the most complex assets, or simply to the production process. In this regard, we would like to extend the word *system* , referred to by Glaessgen and Stargel, to component, part, asset, group, and process:

- Component twins : They represent the *atom* of a digital twin, showcasing the smallest functioning component. An example of a component twin can be a synthetic sensor able to replace a physical sensor given other measures.
- Parts twins : They refer to components of slightly lesser significance but serve a similar purpose. An example of a part twin can be the cooling system of a machine.
- Asset twins : When multiple components collaborate, they create what is termed an asset . Asset twins allow for the examination of interactions among these components, generating extensive performance data that can be analyzed to yield actionable insights. An example of an asset twin can be a digital twin of an engine.
- Unit twins : At a higher level of detail, unit twins illustrate how various assets integrate to form a complete operational system. An example of a system twin can be a wind farm.
- Process twins : They represent the broadest perspective, illustrating

how systems collaborate to operate an entire production facility. An example of a process twin can be a refinery.

Of course, as the system becomes more complex, it is preferable to develop different models for the specific purpose; for example, it is easier to develop a model of a chemical plant to optimize production and one to measure degradation, although obviously, they are not independent.

Implementing a digital twin in IIoT

As we learned in the previous chapter, we can develop an analytic either by using a data-driven approach or by using a physics-based mathematical formula. A data-driven approach tries to infer a statistical or mathematical model from data using advanced technologies, such as ML or DL. A physics-based approach tries to emulate the numerical model of the equipment. Both methods have their own pro and cons.

While the data-driven approach is good for general purposes and can be applied to different equipment models, it is not always able to make decisions about new behavior that has not been acquired from the training data. The physics-based model is very specific to each asset, but it might provide inaccurate results due to the degradation of the equipment or uncalibrated data. Today, the best thing to do is to use a combination of both approaches, instead of using a single model.

In the near future, however, thanks to AI and in particular to multimodal AI / general AI (AGI) (see *Figure 14 .4 ahead*), when we have enough data, even this simplification can be overcome. In other words, as technology evolves, an *Omni digital twin* like the one proposed by Glaessgen and Stargel becomes increasingly possible.

In this section, we learned what a digital twin is. In the next sections, we will develop two models of digital twins, one purely AI-based, and one physics-based. However, before diving into practice, we need to delve a little deeper into the concepts of AI.

AI and IoT

When working with IIoT, we typically need to work with a plethora of technologies, from statistical data modeling using stochastic processes to AI. These technologies often come from other fields, such as trading, image processing, and natural language processing (NLP), and have been adapted for use in IIoT. AI and in particular ML is not part of IoT, but it is strongly linked to it, so much so that a new term was founded: artificial intelligence of things (AIoT).

Today, when we speak about AI, we refer to ML; let's see what ML is.

ML

ML algorithms try to build a model from data. ML algorithms can be supervised learning , unsupervised learning , or reinforcement learning (RL).

Supervised learning

In supervised models, the algorithm learns from a set of data that is labeled with the correct answers. For instance, in the renewable energy sector, we might have a set of data that contains the input of a wind turbine, including the amount of wind and the turbine's position, and the output, which is the amount of power generated. From this data, we can infer a model that is able to predict the output of the wind turbine based on the input. Other examples of supervised learning include regression, classification, tagging, and ranking. Contrastive learning, anomaly detection, and sequence learning can be considered semi-supervised learning because you do not know all the labels, but just a little information.

Unsupervised learning

Unsupervised models don't know the correct answer and try to infer a model by using just the data input. Some examples of unsupervised learning models include clustering, instance-based models, subspace estimations, and generative adversarial networks (GANs).

RL

RL refers to goal-oriented algorithms that learn how to reach a goal. RL models execute a series of steps and receive a reward when the goal is reached. RL has been used in gaming or multi-agent systems. It has also been used recently in IoT to make decisions about which actions should be carried out to optimize production or avoid maintenance. In IIoT, we can use RL to reward the algorithm when it raises the alarm about appropriate events correctly and to penalize it if it raises a false alarm. In the industrial sector, the control room is always ready to check issues and provide feedback when alerted by an RL algorithm.

ML development

In *Chapter 13* , we saw the steps needed to develop a model:

1. Problem definition
2. Data preparation
3. Exploratory data analysis
4. Features engineering
5. Model development
6. Model deployment
7. Monitoring

From an implementation point of view, an AI developer follows the same process, but before developing a model, they must prepare the data. It reserves a percentage for training (from 70% to 80%) and one for validation (from 20% to 40%).

Each model requires specific preprocessing, some of which are listed here:

- Scaling : Data is scaled to a range from 0 to 1.
- Dimensionality reduction : Sometimes, it is appropriate to reduce the number of dimensions to overload the model with redundant information.
- Imputation of missing data : There may be missing data; the best way is to replace them either with a fictitious value, with the average, or with other strategies.
- Converting categorical data to numeric data : AI models are not capable of working with categorical data (for example, *good* or *bad* , *red* or *green*) so it must be converted to numbers.
- Converting text to numerical data : Similar to categorical data, text data must be converted to a number or a vector of numbers (for instance, embedding).
- Rebalancing the dataset : Industrial data are affected by strong imbalances between classes (we have already seen the problem of anomaly detection in *Chapter 13*), which is why it is necessary to adopt mitigation policies. The simplest way is to reduce the samples of the majority class or generate new data for the minority ones.
- Mini batching : The model is iteratively fed with blocks of data.

See the official scikit-learn page for more information: https://scikit-learn.org/stable/data_transforms.html .

Once the dataset is available, the first step is to begin the training phase and then evaluate the performance on the validation test set.

The final goal of an ML model is to develop a model capable of understanding the training set data sufficiently to generalize the behavior in the validation set and in production. A model that is too big will simply memorize the training set and it will not really understand the relationships between input and output, thus, not generalizing in the test set (overfitting). On the contrary, an insufficient model or one trained on a few pieces of data will not allow the development of good analytics (underfitting).

The following figure shows some examples of underfitting versus overfitting. Overfitting obsessively follows the data and underfitting roughly approximates them, while a good fitting is a good balance that will allow the behavior to be generalized even in the test set.

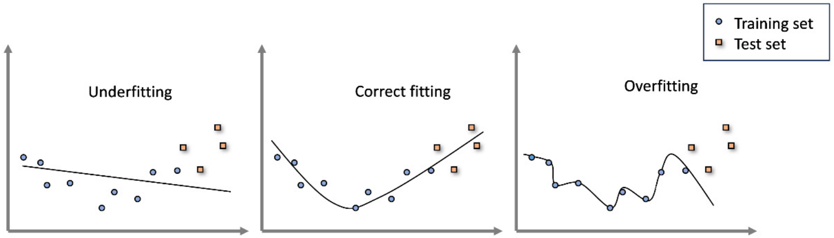


Figure 14.1 – Underfitting versus overfitting

> ### *More about ML*

> *The process of developing ML models is very complex and there are other techniques to efficiently build and maintain an ML model (e.g., hyperparameter tuning, folding, metrics evaluation, unbalanced data management, generative model, reinforcement learning by human feedback, explainable AI, continuous learning and uncertainty quantification). In this section and the next section, we have presented what is essential for understanding the exercises in this chapter (for more details, see Machine Learning with PyTorch and Scikit-Learn: Develop machine learning and deep learning models with Python , Sebastian Raschka, Yuxi (Hayden) Liu, Vahid Mirjalili, Packt Publishing, 2022).*

Scikit-learn

Scikit-learn (https://scikit-learn.org/) is the common framework used to develop classic ML algorithms. Scikit-learn supports more than 100 linear and nonlinear models, classification and regression models, and supervised and unsupervised models. Scikit-learn also supports a string transformation pipeline to convert and preprocess data. From an implementation point of view, scikit-learn is based on a very clear interface and a few methods, which allow you to switch from one model to another and to interchange preprocessors with a few changes. The following table summarizes the most important methods:

Method	Description	
fit(X, y)	Model training	Classification, regression, dimensionality reduction, clustering, and preprocessing
predict(X)	Model evaluation	Classification and regression
fit_transform(X, y)	Preprocessing and data transformation	Dimensionality reduction and preprocessing
transform(X)	Data transformation	Dimensionality reduction and preprocessing
predict_proba(X)	Probability evaluation	Classification

Table 14.1 – Scikit-learn cheat sheet

We will see scikit-learn in action in the next exercises. Scikit-learn is perfect for classic ML but if we want to develop more advanced models, we need DL.

DL

In 2010, the ML technique DL became an important new method for implementing neural networks (NNs). Thanks to the application of a hierarchical distributed representation across multiple layers, NNs have been restored to their former glory.

A NN is a network of multiple neurons fully (or partially) connected to each other. The mathematical function of a single neuron is the weighted sum of the output of the previous neurons (see the following figure); the output is then (optionally) sent to an activation function and sent to the next network layer.

Weights are automatically updated during the training phase; during the first step, we feed the NN with the given input (feed-forward), then we compare the output and the expected output with the output of the NN (the error), and finally, we backpropagate the error, using an optimization method such as gradient descent (GD), to update the weights and the bias of all the neurons.

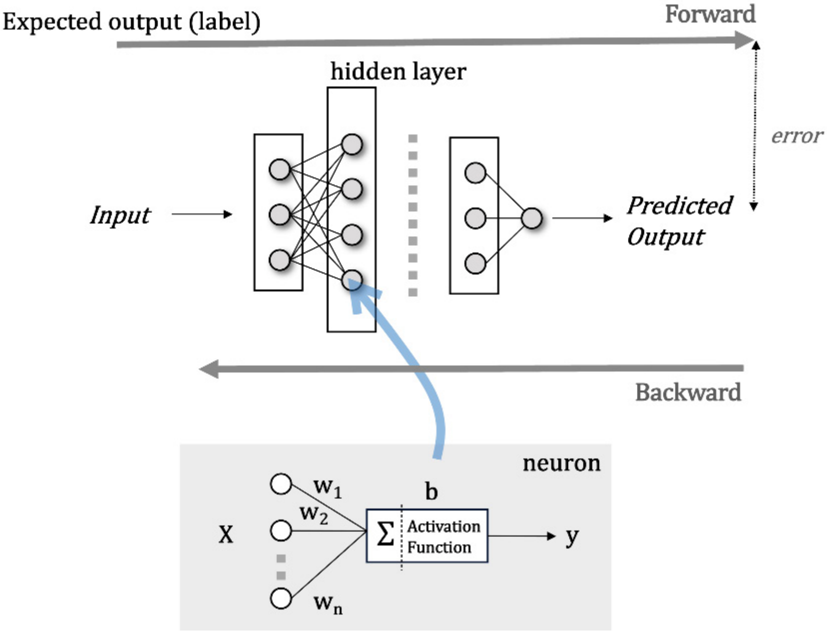

Expected output (label)
Forward
hidden layer
error
Input
Predicted Output
Backward
neuron
X
w1
w2
wn
b
Σ
Activation Function
y

Figure 14.2 – NN and neuron model

From a mathematical point of view, the backpropagation algorithm is based on the GD method, that is, it seeks the minimum error by going in the opposite direction to the gradient; for computational reasons, it is preferable to apply the GD iteratively to small batches of data and for this reason, it is called mini-batch GD or stochastic gradient descent (SGD).

> ### *Mini-batch GD, SGD, and Adam*

> *Formally, **SGD** is the application of GD to a single point, while **mini-batch GD** is the application of GD (and therefore SGD) to a group of points (a mini-batch). Of course, the amount of data (batch size) can contribute to the convergence of the SGD (see also On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima , Keskar et al ICML 2016 (https://arxiv.org/abs/1609.04836)).*

The following figures provide an intuitive explanation of SGD. During the GD process, the weights are updated according to the following formula:

$Wx = Wx - LR (\partial error / \partial Wx)$

As the network is fed with a portion of the data, the result fluctuates significantly, but converges to the minimum:

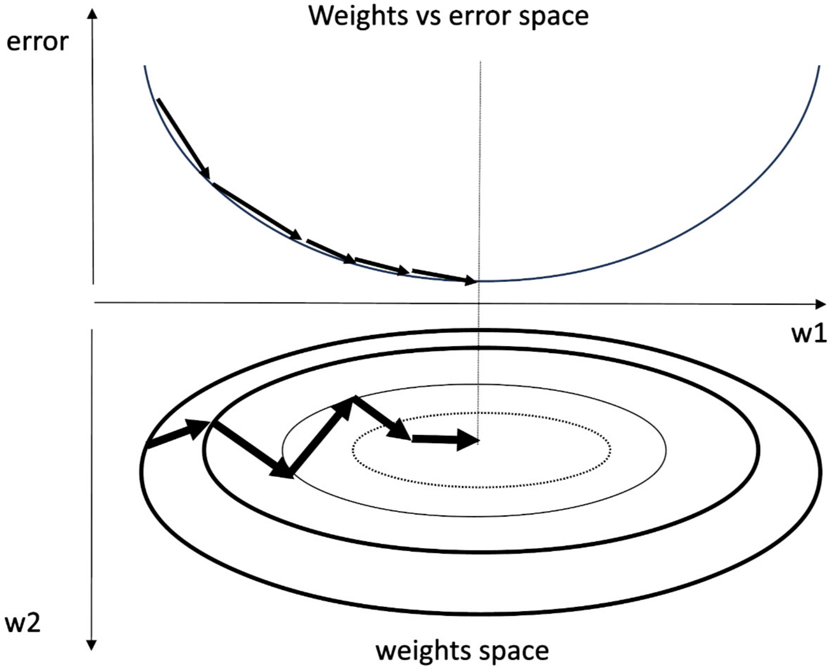

Weights vs error space
error
w1
w2
weights space

Figure 14.3 – Mini-batch GD in action and its natural fluctuation

Since SGD is an iterative method, the descent speed can be controlled by a parameter known as learning rate (LR). The LR is a scalar from 0 to 1; the smaller it is, the more iterations will be necessary; however, too large an LR can cause the algorithm to jump from one point to another without ever reaching the minimum. In order not to worry too much about the LR, an optimizer known as Adam is used; Adam uses some heuristics to optimize the LR and also avoid local minima.

> ### *Optimizers*

> *It is possible to choose from a wide variety of optimizers for the best GD algorithm (Adam, AdamW, Adamax, RSProp, SGD, etc.). In this chapter, we will only use Adam (https://pytorch.org/docs/stable/optim.html).*

We will see a practical implementation later on in the *Developing a digital twin* section.

TensorFlow

TensorFlow (https://www.tensorflow.org/) is an open source library for ML algorithms with great support for GPUs or tensor processing units (TPUs). The easiest way to work with TensorFlow is through Keras and Python, but it is also possible to use other languages (such as C++) directly with TensorFlow.

PyTorch

PyTorch (https://pytorch.org/) is the common framework for developing NNs. PyTorch is extensively used in DL and in particular in generative AI (GenAI). We will use PyTorch in the next exercise to develop our digital twin model.

GenAI

GenAI is a subset of ML and DL. From 2023, thanks to the diffusion of ChatGPT, DALL-E and in particular the large language model (LLM), GenAI has also gained a lot of attention in IIoT. GenAI is capable of generating content from a given instruction (prompting). The first model of GenAI was the variational autoencoder (VAE), then, thanks to GANs, GenAI has achieved a high level of attention from the scientific community; therefore, with the introduction of transformers, stable diffusion and reinforcement learning by human feedback, GenAI has unlocked a new level, which goes toward AGI.

AGI

AGI or strong AI is the next level of AI to achieve human performance. The basic idea of AGI is to build general models (foundation models) able to work on several domains with low effort to retrain the model. Today, humans learn in just a few steps, while current AI requires a lot of data. The basic idea is to develop sufficiently powerful and general models capable of reusing the patterns already acquired and adapting them to new contexts with little data (few-shot learning) or no data (zero-shot learning).

The journey to AGI has just started and we expect to get the first real AGI in 2030. The debate, however, is still ongoing: some researchers think that GPT-4 can be considered AGI, while others say that for AGI, we will have to wait until 2050.

The following figure highlights the main concept covered in this section. AI today (2024) is mainly ML and in particular DL. The next step is to achieve the AGI concepts. Obviously, there are some ethical and security concerns that are beyond the scope of this book. In the next section, we will see how to work with AI in the field of IIoT:

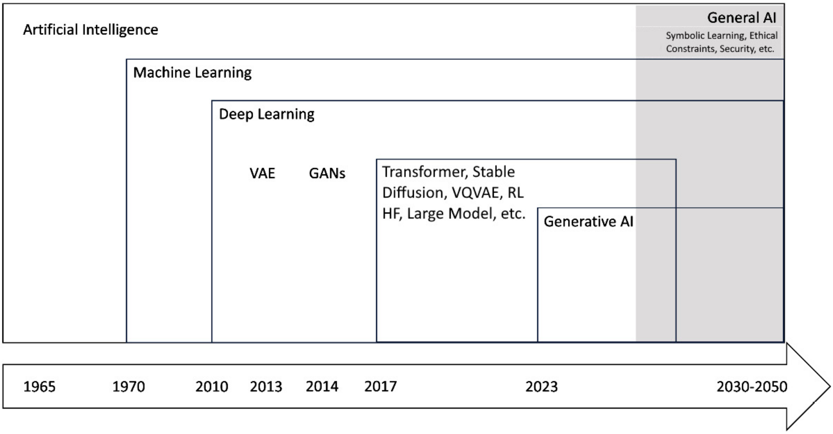

Artificial Intelligence
General AI
Symbolic Learning, Ethical Constraints, Security, etc.
Machine Learning
Deep Learning
VAE
GANs
Transformer, Stable Diffusion, VQVAE, RL HF, Large Model, etc.
Generative AI
1965
1970
2010
2013
2014
2017
2023
2030-2050

Figure 14.4 – A simplified history of AI and the main technologies that have driven its evolution

In the next section, we will learn how to put into practice the concepts just studied, and we will develop our first digital twin.

Developing a digital twin

In this section, we will develop a digital twin with Python, PyTorch, and pandas. Let's begin with the first step.

Preparing the development environment

Let's prepare the development environment. From the command line, install the required packages:

```
$ pip3 install seaborn scikit-learn
pandas matplotlib notebook
$ pip3 install torch torchvision
torchaudio --index-url https://
download.pytorch.org/whl/cpu
```

Then, start the notebook. From the command line, run the following command:

```
$ jupyter lab --allow-root --
NotebookApp.token='' --NotebookApp.
password=''
```

Finally, connect to `http://localhost:8888`.

Alternatively, you can simply start Jupyter Notebook using the Docker image released with the official repository of this book. The code for this chapter is available in the official repository at https://github.com/PacktPublishing/Hands-On-Industrial-Internet-of-Things-Second-Edition .

From the *Chapter 14* directory, build the container and run it. From the command console, execute the following commands:

```
$ docker compose build
$ docker compose up
```

We already studied Jupyter lab, therefore, we are ready to start coding.

Evaluating the RUL of 100 engines

For this exercise, we want to estimate the RUL of a jet engine; we will use the free Turbofan Engine Degradation Simulation Data Set provided by

NASA.

Steps 0 and 1 – problem statement and dataset

The dataset can be downloaded from https://data.nasa.gov/Aerospace/CMAPSS-Jet-Engine-Simulated-Data/ff5v-kuh6/about_data . For our simplified exercise, we will just use the `train_FD001.txt` file. The challenge proposed by NASA is to calculate the RUL of the 100 engines that are listed. The data is provided in a compressed format where the 26 columns are as follows:

- Unit number (the engine ID)
- Time, in cycles
- Operational setting 1
- Operational setting 2
- Operational setting 3
- Sensor measurement 1
- Sensor measurement 2
- …

- Sensor measurement 21

In the dataset, the time series ends sometime before the system failure.

Problem statement

For our exercise, we not only want to calculate the RUL but also predict the upcoming system failure in the last remaining cycles. We assume that one cycle is equal to one day.

Preparing the environment

We need to prepare our environment as explained in the following steps.

Download NASA's dataset in the current folder. In the same directory, create a new Jupyter Notebook, then write the following lines of code to import the data and execute the cell:

```
import pandas as pd
import numpy as np
columns = ['unitid', 'time',
'set_1','set_2','set_3']
```

```
columns.extend(['sensor_' + str(i) for
i in range(1,22)])
df_train = pd.read_csv('./
train_FD001.txt',
delim_whitespace=True, names=columns)
df_test = pd.read_csv('./
test_FD001.txt',
delim_whitespace=True, names=columns)
```

To simplify our exercise, just focus on unit number `1` . In a new cell, please add the following code and execute the cell:

```
unitid = 1
df_train =
df_train[df_train['unitid']==unitid]
df_test =
df_test[df_test['unitid']==unitid]
```

Step 2 – EDA

The dataset is good, so we don't need to clean the data. If we analyze the standard deviation of some variables, however, we can see that **`set_3`** , **`sensor_1`** , **`sensor_5`** , **`sensor_10`** , **`sensor_16`** , **`sensor_18`** , and **`sensor_19`** have a standard deviation of 0, so we can remove them.

The following code performs the standard deviation analysis. In a new cell, please add the following code and execute the cell:

```
df_train.std()
```

Then, we can remove the aforementioned variables, as follows in a new cell; please add the following code and execute the cell:

```
X_columns =['set_1', 'set_2',
'sensor_2', 'sensor_3', 'sensor_4',
'sensor_7', 'sensor_8',
'sensor_9',  'sensor_11', 'sensor_12',
'sensor_13', 'sensor_14',
'sensor_15',  'sensor_17',
'sensor_20', 'sensor_21']
df_train=df_train[X_columns]
df_test=df_test[X_columns]
```

We can now analyze the correlation between the sensors. In a new cell, please add the following lines of code and run the cell:

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.pairplot(df, vars=X_columns)
```

```
plt.show()
```

The result should appear like the following figure:

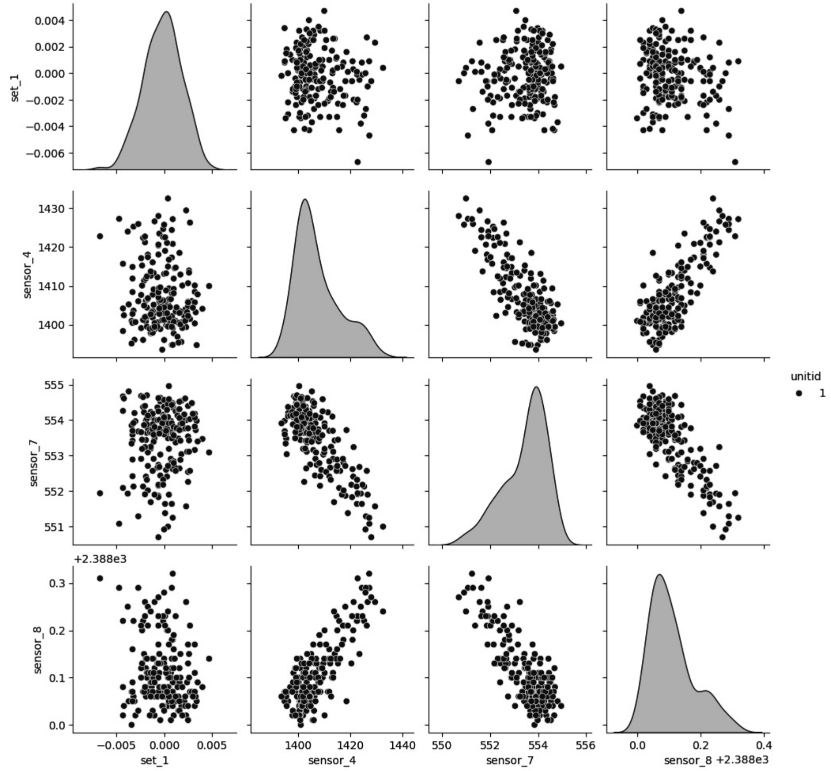


Figure 14.5 – Pair plot visualization for only five engines and five variables

We can see that some sensors are correlated (**`sensor_4`** versus **`sensor_8`**) or anti-correlated (**`sensor_4`** versus **`sensor_7`**); in fact, they align on a diagonal. We could decide to keep only **`sensor_4`**

, however, we will let the model use all the information.

We can also see the trend of the first five engines:

```
import matplotlib.pyplot as plt
import seaborn as sns
i=0
for column in df1:
    if ('sensor' in column):
        i=i+1
        ax= plt.subplot(5,5,i)
        ax = sns.lineplot(x="time",
y=column, hue='unitid', legend=False,
data=df, ax=ax)
```

The result should look as follows:

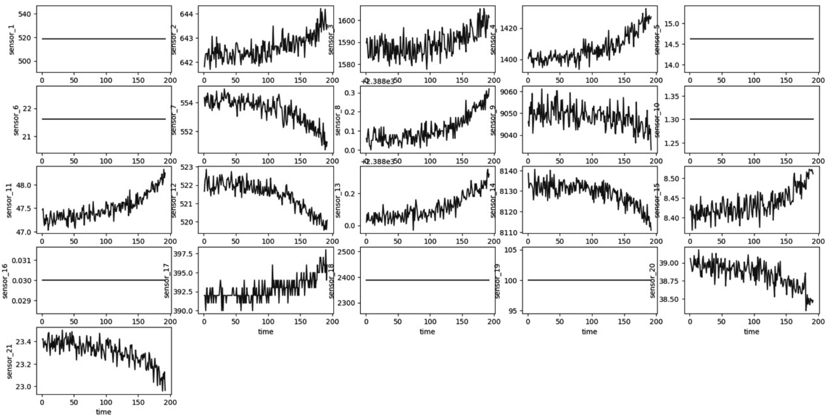


Figure 14.6 – Time series of five engines

Please note the decaying or increasing trend close to data sample #100.

Step 3 – building the model

Let's build our algorithm using the steps described previously.

Building the dataset

We have no information on how the asset degrades; the only thing we know is that the last point in the training set is the cycle before the break. So, as a label, we assume a linear degradation starting from 40% of the dataset. This assumption is supported by the trend of the data that seems to degrade from a certain point onward (see, for example, the trend of **`sensor_21`**).

In a new cell, please add the following lines of code and run the cell:

```
def create_train_dataset(dataset):
    dataX, dataY = [], []
    n = len(dataset)
    start=int(len(dataset)/10*4)
    for i in range(start):
        a = dataset[i]
        b = (n-start) / n
        dataX.append(a)
        dataY.append(b)
    for i in range(start, n):
        a=dataset[i]
        b=(n-i) / n
        dataX.append(a)
        dataY.append(b)
    return
np.array(dataX).astype('float32'),
np.array(dataY).astype('float32')
```

Now, we can create our training set and test set. In the previous cell, please add the following lines of code and run the cell:

```
X_train,y_train
=  create_train_dataset(df_train.values)
n = len(X_train)
```

We cannot make any assumptions about `y` of the test set; we only know that the last point has an RUL equal to `112` as reported by NASA data. So, in a new cell, build only `X` of the test set:

```
X_test,_
=  create_train_dataset(df_test.values)
```

Selecting variables

Normally, we split the dataset so that we use 70% for training and 30% for testing. In this case, however, we don't need to do this. The dataset provided by NASA has four files that have the **`train_`** prefix and four files that have the **`test_`** prefix. We can use the train files for training and the test files for validation, as suggested by NASA.

Defining the model

We want to develop a digital twin based on previous data. The model will be based on an NN using PyTorch. The basic steps for building our model are as follows:

1. Scaling the data set in the interval **`[0,1]`** .
2. Building the NN.
3. Training the model.
4. Testing the model.

To scale the dataset between `0` and `1`, we can use scikit-learn's MinMax Scaler. In a new cell, please add the following lines of code and run the cell:

```
from sklearn.preprocessing import
MinMaxScaler
def scale_dataset(ds_train, ds_test):
    scaler =
MinMaxScaler(feature_range=(0, 1))
    ds_train =
scaler.fit_transform(ds_train)
    ds_test =
scaler.transform(ds_test)
    return ds_train, ds_test
X_train, X_test =
scale_dataset(X_train, X_test)
```

Scaling is necessary so that the data are all in the same range; for example, if a variable varies between 0 and 100 and 1 between 0 and 100, the SGD will tend to more easily chase the errors determined by the variable with the widest excursion. Scaling helps us to have a more equitable attitude toward all the variables. It is therefore good practice in the development of NNs to scale the input data.

Now, we can build the model: the first layer has 16 neurons and expects 4 input variables. The second hidden layer has 32 neurons. The output layer has one neuron, which predicts the output. In a new cell, please add the following lines of code and run the cell:

```
import torch.nn as nn
class JetDigitalTiwnModel(nn.Module):
    def __init__(self, input_size,
hidden_size=16):
        super().__init__()
        self.hidden1 =
nn.Linear(input_size, hidden_size)
        self.act1 = nn.ReLU()
        self.hidden2 =
nn.Linear(hidden_size, hidden_size*2)
        self.act2= nn.ReLU()
        self.out =
nn.Linear(hidden_size*2, 1)
        self.act = nn.ReLU()
    def forward(self, x):
        x = self.act1( self.hidden1
(x) )
        x = self.act2( self.hidden2
(x) )
        x = self.act( self.out(x) )
        return x
```

During compilation, PyTorch will use the best numerical library. The backend will automatically choose the best way to model the NN to run on your hardware. This might be with CPU, GPU, or distributed computation.

The next step is to train the model. In a new cell, please add the following lines of code and run the cell:

```
import torch
from torch.utils.data import Dataset
class MyDataset(Dataset):
    def __init__(self, x, y):
        super(MyDataset,
self).__init__()
        self.x = x
        self.y = y
    def __len__(self):
        return self.y.shape[0]
    def __getitem__(self, index):
        return self.x[index],
self.y[index]
```

Now, define the train function. In a new cell, please add the following lines of code and run the cell:

```
def train_model(model, X_train,
y_train, num_epochs = 1000,
learning_rate = 1e-4, batch_size=64):
    X_train = torch.tensor(X_train,
dtype=torch.float32)
    y_train = torch.tensor(y_train,
dtype=torch.float32).reshape(-1,1)
    # mean-squared error for
regression
    criterion = torch.nn.MSELoss()
    optimizer =
torch.optim.Adam(model.parameters(),
lr=learning_rate)
    loader =
torch.utils.data.DataLoader(MyDataset(X_tra
y_train), batch_size=batch_size,
shuffle=True)
    history = {'loss':[]}
    # Train the model [0]
    for epoch in range(num_epochs):
        for batch_idx, (_X, _y) in
enumerate(loader):
            model.train()
            outputs = model(_X) # [1]
            optimizer.zero_grad()
            loss = criterion(outputs,
_y) # [2]
            loss.backward() # [3]
            optimizer.step() # [4]
        if epoch % 100 == 0:
            print("Epoch: %d, loss:
%1.5f" % (epoch, loss.item()))
```

```
            history['loss'].append(loss.ite
    return model, history
```

Before continuing, let's analyze the code just written. Notice the two loops **[0]** that gradually extract the mini batches of data as we learned in the *DL* section. Then, the data is applied (**[1]** forward, see *Figure 14 .1*), then the error **[2]** is calculated with the root mean squared error (RMSE) function and the weights are updated using the backward function **[3]** . Finally, optimization is called to calculate the new LR **[4]** .

Let's instantiate the model and start training. In a new cell, please add the following lines of code and run the cell:

```
model =
JetDigitalTiwnModel(input_size=X_train.shap
model, history = train_model(model,
X_train, y_train)
```

Finally, evaluate the results. In a new cell, please add the following lines of code and run the cell:

```
model.eval()
    with torch.no_grad():
        out_data =
model(torch.tensor(X,
dtype=torch.float32))
rul = (out_data*n).numpy().flat[-1]
```

We have estimated a RUL of 113 cycles. The expected answer was 112 cycles, so we have an error of about 1%.

Before concluding our exercise, let's analyze the loss. In a new cell, please add the following lines of code and run the cell:

```
import matplotlib.pyplot as plt
plt.plot(history['loss'])
```

The expected loss should be similar to the following figure:

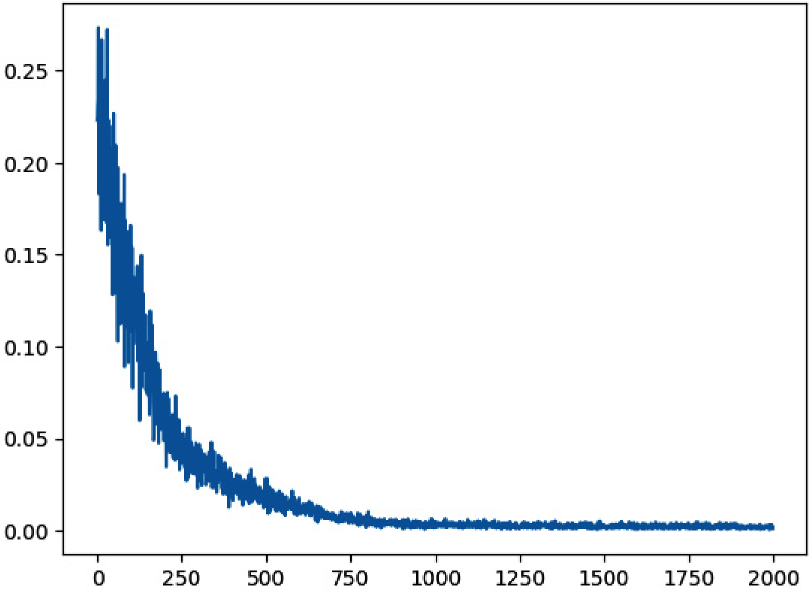


Figure 14.7 – Loss during training

The graph shows how the algorithm is learned; in fact, the error gradually decreases, but note the typical behavior of the SGD, which continuously oscillates until it converges. This is the typical behavior that we expected and that we studied in the *DL* section.

Comparing the model with the baseline model

It is good practice to try a simple model to compare it to a complex model. The simplest model is the dummy mean or a linear regression. In this exercise, we will use a model called Gaussian process regression (GPR) as the baseline model. GPR uses the Bayesian approach, which is very robust to noisy data. We will use the implementation of the scikit-learn package. The implementation is relatively simple:

1. Define the model.
2. Train the mode by calling the `fit` method.
3. Evaluate the model by calling the `predict` method.

Open a new cell and add the following lines of code:

```
import sklearn
from sklearn.gaussian_process.kernels
import RBF, WhiteKernel
from sklearn.utils import shuffle
```

Define the kernel of GPR and the model:

```
kernel = 1 * RBF(length_scale=1.0)
baseline_model =
sklearn.gaussian_process.GaussianProcessReg
```

Then, train the model with the previous data:

```
baseline_model.fit(X_train, y_train)
```

Now, let's evaluate the outcome. In a new cell, run the following lines of code:

```
y_pred =
baseline_model.predict(X_test)
y_pred = np.array(y_pred * n)
print( y_pred[-1] ) #attended 112
```

The output is **`117`**, which is quite close to the expected **`112`**.

Step 4 – packaging and deploying

We can now develop and use the unit test and the regression test to check the quality of our analytics.

To deploy analytics on the cloud, we have to decide whether to use cold-path or hot-path. The developed algorithm needs a lot of data and should run daily or weekly so that we don't need to use a stream-based infrastructure and we can use the batch processing mechanism.

Step 5 – monitoring

It is a good idea to review the performance of the algorithm quarterly, calculating the false positives and false negatives with the newly acquired datasets.

Monitoring a wind turbine

In this simple exercise, we want to monitor the degradation of a wind turbine over a period of a few years. We already know the physical model of a turbine and we also know that the efficiency of a wind turbine can decrease linearly by about 16% every 10 years (see *How does wind farm performance decline with age?* , Iain Staffell, Richard Green, Renewable Energy, Volume 66, 2014).

Steps 0, 1, and 2 – problem statement, dataset, and exploratory data analysis

We want to evaluate the RUL, assuming that we need to replace the wind turbine after it loses 16%

of its performance. The dataset is a CSV file with a year's worth of data. The dataset contains the following:

- Time (sampled every 10 minutes) Wind speed (m/s)
- Generated power (kW)

We can compare our data with the physical model of a turbine to evaluate the loss of performance in order to predict degradation in the next 10 years.

Step 3 – building the model

In this case, we want to implement a physics-based approach by building a mathematical relationship between the wind speed (m/s) and the amount of power generated (kW). The following Python code implements our model. Open a new Jupyter Notebook and write in the first cell the following lines of code:

```
def wind_turbine_model(x):
    if x<=4.5 or x>21.5:
        return 0
    return 376.936 - 195.8161*x +
33.75734*x**2 - 2.212492*x**3 +
0.06309095*x**4 - 0.0006533647*x**5
```

The following is a plot of the model with varying wind speeds on the *x* -axis:

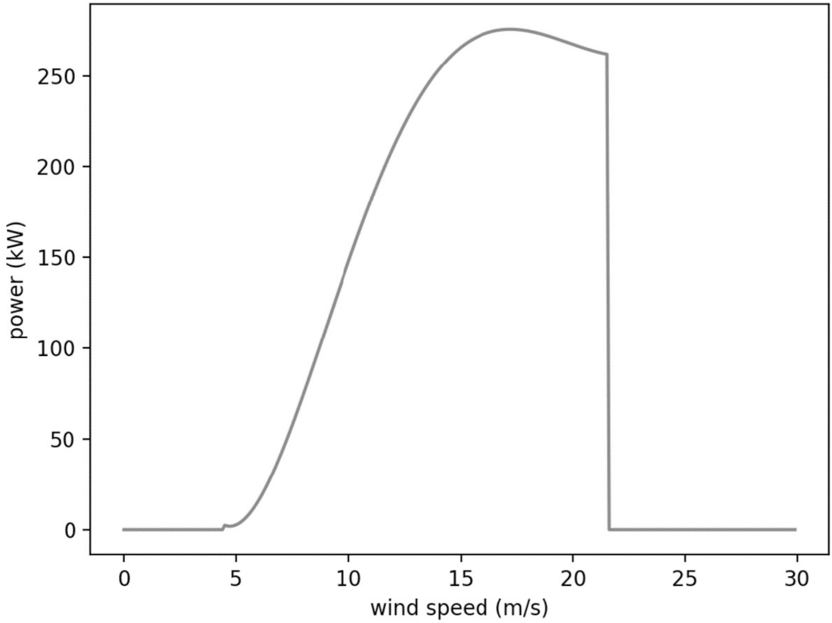


Figure 14.8 – A model of a wind turbine (wind speed versus generated power)

Note that this digital twin is only valid for a specific wind turbine model, and we cannot generalize it to another. Generally speaking, however, every wind turbine will follow a similar sigmoidal model. Let’s build our algorithm.

Import the dataset and calculate the expected power versus the measured power. After that, visualize the dataset. In a new cell, please add the following lines of code and run the cell:

```
import pandas as pd
import matplotlib.pyplot as plt
# load data
df = pd.read_csv('./wind_turbine.csv')
expected_power =
[wind_turbine_model(x) for x in
df.wind_speed_ms]
reference_power =
[wind_turbine_model(x) for x in
range(0,30)]
# show data and reference
plt.plot(df.wind_speed_ms,
df.power_generated_kw,'.r')
plt.plot(reference_power,'k')
plt.show()
```

The following diagram shows the output of the algorithm:

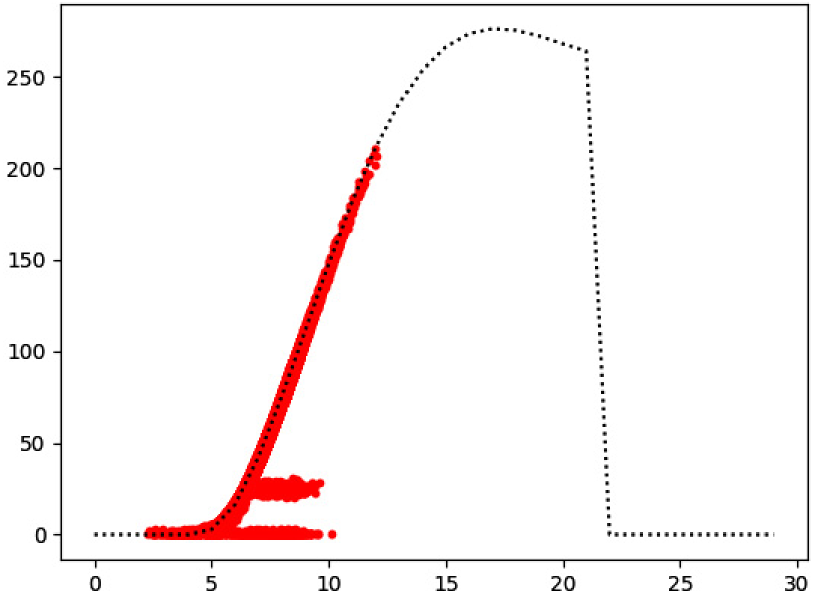


Figure 14.9 – The dataset compared with the reference model

We can now evaluate the loss of power using the following equation: **`(measured_power - expected_power) / expected_power`**.

In a new cell, please add the following lines of code and run the cell:

```
de=[]
ts=[]
for i in range(0,len(expected_power)):
    mp = df.power_generated_kw[i]
    ep = expected_power[i]
    if ep>0 and mp>0:
        t = df.cycle_10_mins[i]
        de.append( (mp-ep) / ep )
        ts.append(t)
```

Finally, to predict the degradation of the equation over the next nine years, we can estimate the first-order equation of the degradation using the following code. Add a new cell, add the following lines of code, and run the cell:

```
samples=365*24*6
year=9
import numpy as np
z = np.polyfit(ts, de, 1)
print("degradation in %s years will be
%.2f %%" % (year,
100*(z[0]*(samples*year) + z[1])) )
```

The output degradation in 9 years will be -15.72%. This is not the same as the expected – 16%, but it is quite close.

Steps 4 and 5 – packaging, deploying, and monitoring

We can apply the same considerations as in the previous exercise in this case.

We have seen some practical examples of digital twins; in the next section, we will explore some common platforms.

Platforms for digital twins

A digital twin's platform for IIoT is not that different from the analytics platforms studied in the previous chapters. All it needs is sufficient storage support to save the status of the model and to manage the life cycle of the digital twin. In other words, digital twins require the model to be tuned initially. They also require the real asset and the digital asset to be synchronized periodically, as shown here:

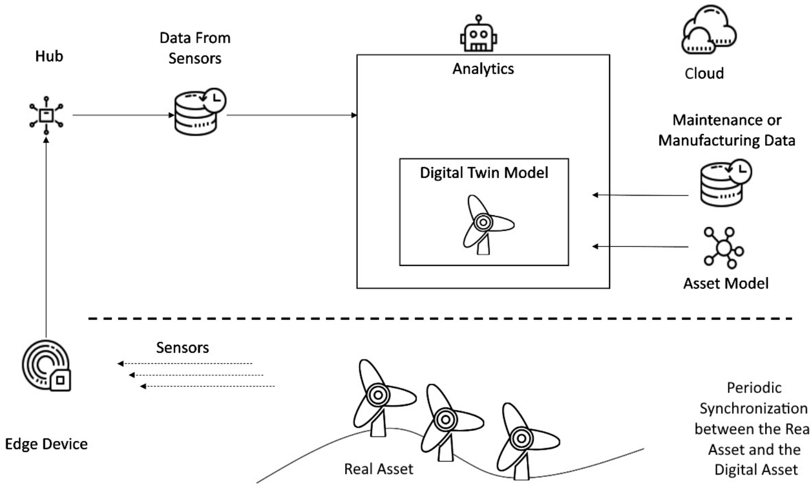

Hub
Data From Sensors
Analytics
Cloud
Maintenance or Manufacturing Data
Digital Twin Model
Asset Model
Sensors
Edge Device
Real Asset
Periodic Synchronization between the Real Asset and the Digital Asset

Figure 14.10 – The components of digital twins

In the next sections, we will see some commercial platforms.

Digital twins platforms

There are many platform s that can support digital twins today. The following is a short list:

- AWS SageMaker : AWS recommends that we use SageMaker as the main platform for ML. With SageMaker, we can define our model to train parameters and hyperparameters. We can also store the model to deploy it later either on the cloud or on-premises. The basic idea of SageMaker is to train the model that is coding it on Jupyter and later deploy the model as a microservice to get the results during normal operation. A high level of computation is required during training.

- Microsoft Azure : Azure proposes a full platform to define digital twins (see https://azure.microsoft.com/en-us/products/digital-twins).
- Google Cloud Platform (GCP): This is one of the best players to build ML.
- NVIDIA Omniverse : This is a platform for industrial applications and leverages OpenUSD for 3D and digital twin workflows. To access NVIDIA Omniverse, go to https://www.nvidia.com/en-us/omniverse/ .
- IBM Cloud : IBM Cloud is the platform offered by IBM that supports digital twins and IoT (see https://developer.ibm.com/articles/digital-twins-and-the-internet-of-things/).
- Ansys : Ansys is a strong simulation environment that is very popular in the industry (https://www.ansys.com/).

Advanced modeling

We are likely to need a more advanced modeling tool in areas such as thermodynamics, mechanics, fluid dynamics, chemistry, transportation, construction, and physics in general. MATLAB (https://www.mathworks.com/products/matlab.html), or Simulink, is the most common commercial software available, but other open source technologies, such as OpenModelica (https://

www.openmodelica.org/), SageMath (http://www.sagemath.org/), or GNU Octave (https://www.gnu.org/software/octave/), can be considered as well. OpenModelica, in particular, supports a Python API, so the physics-based model can be built by an expert with OpenModelica and we can then decide to integrate the model into our analytics at a later date.

In this section, we have discussed the most common digital twin and modeling platforms. However, to develop a complete digital twin, we often need a lot of other information (for example, installation date, aging, and maintenance events), apparently, that is not specific to IIoT.

Other kinds of IIoT data

In IIoT, we frequently speak about time series and sensor data. In some circumstances, however, we can also work with unstructured data. This might include images, such as pictures of geological layers, inspection images, and high-frequency data, or log files, such as equipment shutdown files, security report files, and maintenance events. This data should be considered in the data flow. In fact, as we have already discussed in the first paragraph of this chapter, digital twin models will include all multimodal data; for example, some researchers propose leveraging LLM models to integrate textual data and time series in order to analyze, in a single solution, anomaly or maintenance data.

Summary

In this chapter, we have explored one of the hottest topics of IIoT – the digital twin. We also studied DL and NN. We also explored what the future of IIoT and AI could be.

Next, we learned about the differences between the physics-based and data-driven approaches, and we applied these to two use cases – wind turbines and engines. Finally, we understood the classes of digital twins and how we can implement them with practical examples.

In the next chapter, we will apply the exercises that we have carried out in the past two chapters to cloud-based technology using Azure and AWS.

15

Deploying an Analytics Model

In the previous chapters, we looked at the differences between distinct types of analytics. We implemented some examples of prognostic and diagnostic analytics in the world of the Industrial Internet of Things (IIoT). We also studied how to deploy our analytics on the most common platforms and how to use open source technologies.

In this chapter, we will finalize our exercise by delivering the algorithms developed in *Chapter 14* . We want to highlight the most important challenges to building and deploying analytics in production. We will discover the most important cloud platforms and how they provide a computational infrastructure for training and a service-oriented platform for using the analytical model.

In this chapter, we will explore the following topics:

- Deploying digital-twin analytics using the Azure Machine Learning (Azure ML) service
- Understanding analytics on Azure IoT Edge
- Deploying analytics using Amazon SageMaker
- Understanding Google Cloud Platform (GCP) analytics and multi-cloud solutions

Technical requirements

In this chapter, we need the following prerequisites:

- Docker: https://www.docker.com/
- Git: https://git-scm.com/downloads
- Python 3.9+: https://www.python.org/downloads/

The code for this chapter is available at the official repository at https://github.com/PacktPublishing/Hands-On-Industrial-Internet-of-ThingsSecond-Edition .

Understanding model-as-a-service

In *Chapters 13* and *14* , we studied how to develop physics-based or data-driven analytics (machine learning (ML) or deep learning (DL) and, in general, artificial intelligence or AI). In this final chapter, we will see some analytics deployment strategies.

Once developed, analytics reduced to its most extreme essence is nothing more than a function that converts the input (**`X`**) into an output (**`y`**). Whatever the input – an image, text, or a time series – 90% of analytics can be represented by a function **`f`** such that the following applies:

```
y = f(X)
```

If you use TensorFlow, Keras, or scikit-learn, this function is implemented with the **`predict`** function; with PyTorch, it is the **`__call__`** function.

In *Chapter 8* , we studied the concepts of software-as-a-service (SaaS) and microservices. The SaaS model has become increasingly prevalent thanks to its remarkable flexibility, scalability, and cost-effectiveness, making it a popular choice for businesses of all sizes.

We can thus apply this concept to analytics, specifically to ML models. The foundation of the model-as-a-service (MaaS) concept involves transforming the model into a web service that can be accessed by other business services.

From an implementation point of view, it is enough to create a REST API, using the most common technologies (Flask, FastAPI, NGINX, ...) that loads the model and calls the prediction function.

> ***Tip – watch out for costs or wasted resources***

> *A MaaS architecture is very efficient if each model is used at 100%, but if you have models that are rarely called (for example, monthly reports or analyses), apply a service shutdown policy to avoid costs or wasted resources.*

We have grasped the concept of MaaS, which will guide us through this chapter. Next, we will explore how to deploy a model using two widely used open source packages, Amazon Web Services (AWS), and Azure. As illustrated in the following diagram, the initial step (*1*) involves an AI/ML engineer or data scientist utilizing a dedicated notebook-based studio to test the model and monitor experiments:

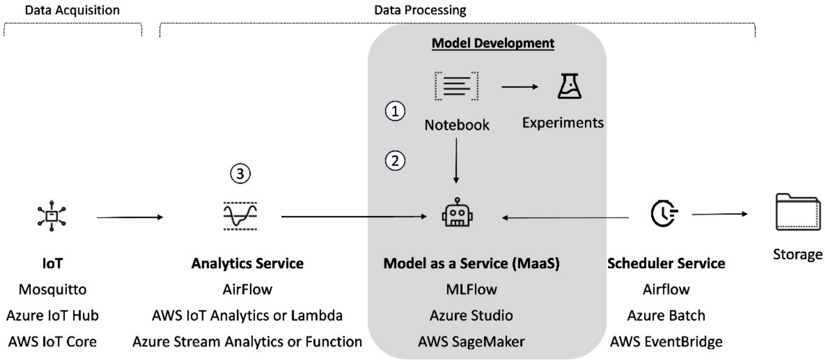

Data Acquisition
Data Processing
Model Development
①
Notebook
Experiments
②
③
IoT
Mosquitto
Azure IoT Hub
AWS IoT Core
Analytics Service
AirFlow
AWS IoT Analytics or Lambda
Azure Stream Analytics or Function
Model as a Service (MaaS)
MLFlow
Azure Studio
AWS SageMaker
Scheduler Service
Airflow
Azure Batch
AWS EventBridge
Storage

Figure 15.1 – Model development in IIoT

Once the model meets the desired performance criteria, it can be deployed as a service (*2*).

Ultimately, we will connect to the IoT data flow via the Stream Analytics service (*3*). AWS and Azure offer complementary services: AWS provides AWS IoT Analytics and SageMaker, while Azure features Azure ML Studio and Azure Stream Analytics. Alternatively, we can leverage on custom AWS Lambda or Azure Functions. Having previously examined these technologies, we will now delve into the deployment of models as a service in the upcoming sections.

In the next section, we will see practical deployment using two popular open source packages, MLflow and Airflow.

Developing our first MaaS with MLflow

In this section, we will (re)develop the digital twin of our wind turbine seen in *Chapter 14* using real data that you can find in the official repository of this book at https://github.com/PacktPublishing/Hands-On-Industrial-Internet-of-ThingsSecond-Edition . Then, we will deploy it using a popular ML operations (MLOps) platform called MLflow (https://mlflow.org). MLflow, based on Python, was born as an experiment tracking platform and extends its capabilities to model storage and serving.

Starting MLflow

To start MLflow, from the command console, install and start the service as shown here:

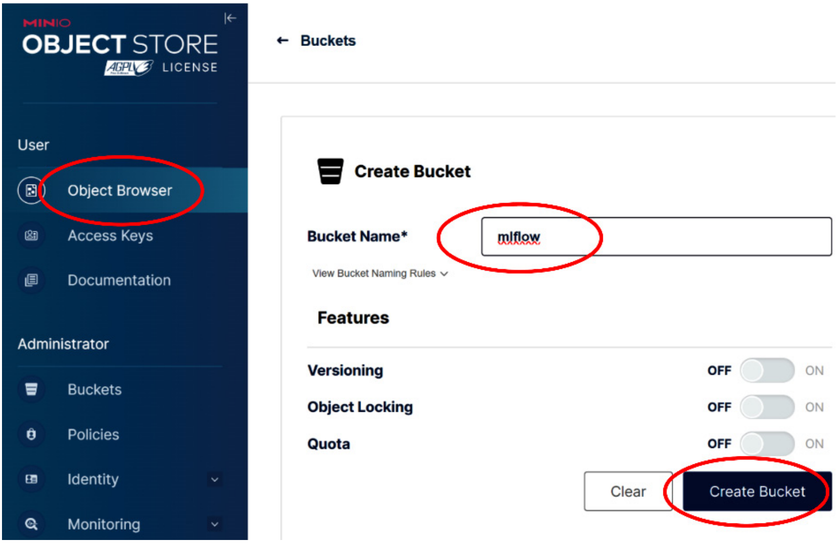
```
$ pip install mlflow
$ mlflow server --host 127.0.0.1 --port 8080
```

Now, you can connect to `http://localhost:8080` to see the web interface.

However, using MLflow as a standalone service is very limiting because its architecture is a bit more complex. MLflow, in fact, requires some services such as Simple Storage Service (S3) and Azure SQL Database to record metadata and models. Therefore, we recommend using Docker Compose, provided with the repository of this book. Thus, on the cloned directory, from the command line, execute the following command:

```
$ cd Chapter15/MLflow
$ docker compose --env-file config.env
up -d --build
```

Now, we need to perform a manual (`init`) operation, at least for the first time, to set up the bucket: connect to the MinIO UI through `http://localhost:9001/buckets` with username `minio_user` and password `minio_pwd` and add a `mlflow` bucket, as shown in the following figure:


MINIO
OBJECT STORE
AGPLv3 LICENSE
User
Object Browser
Access Keys
Documentation
Administrator
Buckets
Policies
Identity
Monitoring
← Buckets
Create Bucket
View Bucket Naming Rules
Features
Clear
Create Bucket

Bucket Name* mlflow

Versioning OFF

Object Locking OFF

Quota OFF

Figure 15.2 – Adding mlflow bucket to MinIO

The service will also start a Jupyter server, accessible at `http://localhost:8888` .

Developing our ML model

In this exercise, we want to develop a simple digital twin of a wind turbine using the dataset already explored in *Chapter 14* . The code of this exercise is available at the official repo.

Step 1 – Problem statement and dataset

In Jupyter Notebook, create a new notebook, and on the first cell, install the MLflow package. Thus, on the first cell, copy and execute the following command.

```
pip install mlflow
```

On a new cell, load the dataset. Add a new cell and execute the following code:

```
import pandas as pd
df = pd.read_csv("wind_turbine.csv")
```

Step 2 – Exploratory data analysis (EDA)

On a new cell, we can show our data. In the same Jupyter notebook, add a new cell, write the following code, and execute the cell:

```
df.plot.scatter(x='wind_speed_ms',
y='power_generated_kw')
```

Step 3 – Building the model

Now, it's time to build our model, track our experiment on MLflow, and register and serve the model.

On a new cell, add the following code to import the required packages:

```
import pandas as pd
import mlflow
import mlflow.sklearn
from sklearn.ensemble import
GradientBoostingRegressor
from sklearn.metrics import r2_score
from sklearn.model_selection import
train_test_split
from datetime import datetime
registered_model_name="wind_turbine"
n_estimators=100
```

Now, we can track our experiment using the MLflow API. On the same cell, add the following code. As you can see, MLflow starts a new experiment and logs some information:

```
with mlflow.start_run() as run:
    mlflow.log_metric("num_samples",
df.shape[0])
    mlflow.log_metric("num_features",
df.shape[1] - 1)
```

Now, we can prepare the dataset. On the same cell, add the following code:

```
    #Split train and test datasets
    train_df, test_df =
train_test_split(
        df,
        test_size=0.3,
    )
    # build X,y and train,test
datasets
    y_train =
train_df.pop("power_generated_kw")
    X_train = train_df.values
    y_test =
test_df.pop("power_generated_kw")
    X_test = test_df.values
```

We are ready to build our model. We will use a simple Gradient Boosting model implemented by scikit-learn. On the same cell, add the following code:

```
    reg = GradientBoostingRegressor(
        n_estimators=n_estimators
    )
    reg.fit(X_train, y_train)
```

Now, we can track our R2 score metrics on MLflow with the following code. On the same cell, add the following code:

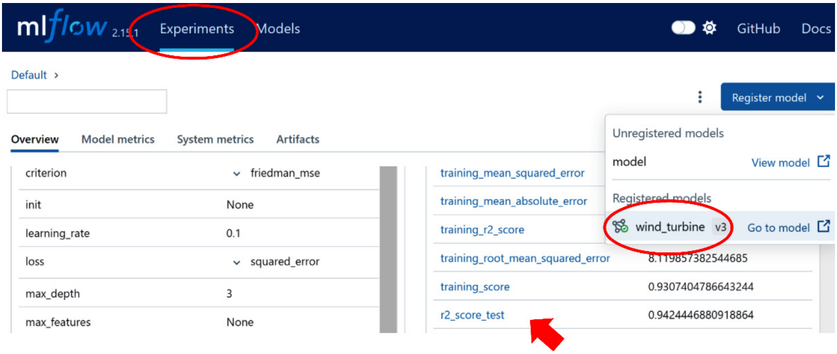
```
    # test the model
    y_pred = reg.predict(X_test)
    mlflow.log_metric("r2_score_test",
r2_score(y_test, y_pred))
```

Finally, register the model in the MLflow registry for serving later:

```
mlflow.sklearn.log_model(
    sk_model=reg,
    registered_model_name=registered_mo
    artifact_path=registered_model_name
)
```

Now, it’s time to execute our cell.

If everything is concluded successfully, all that remains is to access the MLflow UI (**`http://localhost:8080`**) and view our metrics as shown in the following figure:


mlflow 2.15.1
Experiments
Models
GitHub
Docs
Default ›
Register model
Overview
Model metrics
System metrics
Artifacts
Unregistered models
model
View model
Registered models
wind_turbine v3
Go to model

criterion	friedman_mse
init	None
learning_rate	0.1
loss	squared_error
max_depth	3
max_features	None

training_mean_squared_error	
training_mean_absolute_error	
training_r2_score	
training_root_mean_squared_error	8.119857382544685
training_score	0.9307404786643244
r2_score_test	0.9424446880918864

Figure 15.3 – UI of MLflow

As you can see, we reached a 94.2% R2 score, which is acceptable, so we can serve our model.

Step 4 – Packaging and deploying

After the model has been developed and validated, it can now be deployed to serve its intended purpose of processing inputs and generating outputs based on what it has learned.

To do that, we can use MLflow's model server capabilities. This feature allows us to build custom Docker containers, deploy models on Amazon SageMaker, build custom target containers, or deploy as a local server. We will leverage local server capability.

Dependencies

> *MLflow's model server requires some additional packages, such as* **`pyenv`**, **`git`**, *and* **`boto3`**. *We suggest using the Docker image provided with the official Git repository of this book to avoid complex configuration.*

Serving with MLflow API and FastAPI

You can use the MLflow API. On a new cell, add the following line:

```
import mlflow
logged_model = f'runs:/{run_id}/
wind_turbine'
loaded_model =
mlflow.pyfunc.load_model(logged_model)
loaded_model.predict(X_test)
```

Then, we can build our Docker container with FastAPI. From the command console of Jupyter Notebook, install the following packages:

```
pip install "fastapi[standard]" mlflow
```

Now, we can write a new **`main.py`** file with the following code:

```
from fastapi import FastAPI
import mlflow
import pandas as pd
import numpy as np
app = FastAPI()
@app.post(„/predict")
def predict(X_test:float) -> list:
    logged_model = f'models:/
wind_turbine/1'
    loaded_model =
mlflow.pyfunc.load_model(logged_model)
    return
list(loaded_model.predict(np.array([X_test]
1)))
```

Now, it’s time to serve the model and test it. From a Jupyter command console, start the server:

```
fastapi dev main.py --host=0.0.0.0 --
port=8801
```

On another Jupyter command console, we can test it:

```
curl -X 'POST' \ 'http://
localhost:8801/predict?
X_test=9&logged_model=models%3A
```

```
%2Fwind_turbine%2F1' \ -H 'accept:
application/json' \ -d ''
```

The response should be similar to `106.12` .

> ***Tip***

> *How can Jupyter Notebook find the MLflow server? In the* `docker-compose.yml` *file, we have set up the following URI:* `MLFLOW_TRACKING_URI=http://mlflow_server:5000` *. Alternatively, you can set it up explicitly via the* `mlflow.set_tracking` *API (* `MLFLOW_TRACKING_URI` *).*

Serving the model with the MLflow server

Instead of developing our service, we can just use MLflow's server capabilities.

From the command console, connect to the MLflow server using `docker compose exec` to access the MLflow environment:

```
$ docker compose --env-file config.env
exec mlflow_server /bin/bash
```

Then, we can serve the model using the following command:

```
mlflow models serve -m models:/
wind_turbine/latest -h 0.0.0.0 -p 5001
```

Finally, we can test it with the following command:

```
$ curl http://127.0.0.1:5001/invocations
-H "Content-Type:application/ json" --
data '{"inputs": [[9]]}'
```

The expected output is as follows:

```
{"predictions": [105.13453246004944]}
```

We now understand that with MLflow, we can record our experiments and save and serve models, but how can we integrate it into our pipeline?

Understanding Airflow integration

In *Chapter 8* , we utilized Airflow's orchestration capabilities to implement our analytics. Now that we have a clear understanding of the MaaS concept, we can leverage MLflow's functionalities to serve analytics by simply developing an operator that can invoke MLflow through its REST API.

In this context, the architecture outlined in *Chapter 8* has been enhanced with two new components tailored for AI/ML. As illustrated in the following figure, Jupyter Notebook is utilized for model development, while MLflow is employed for experiment tracking and model serving:

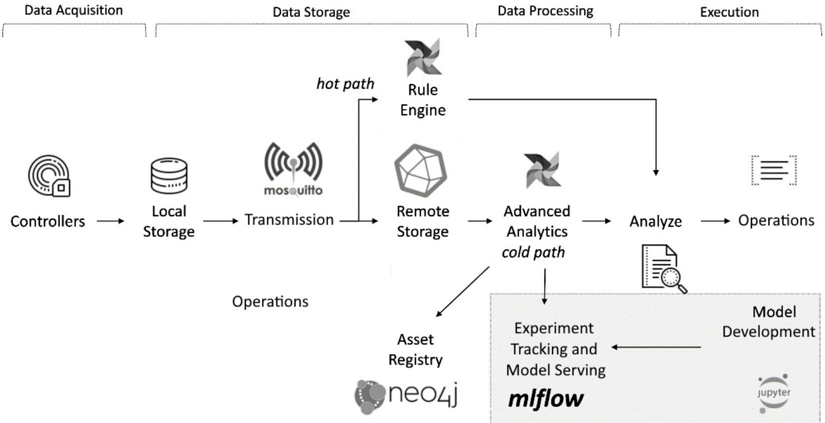

Data Acquisition
Data Storage
Data Processing
Execution
hot path
Rule Engine
Controllers
Local Storage
mosquitto
Transmission
Remote Storage
Advanced Analytics
cold path
Analyze
Operations
Operations
Asset Registry
neo4j
Experiment Tracking and Model Serving
mlflow
Model Development
jupyter

Figure 15.4 – On-premises architecture proposed in Chapter 8 with two additional components for AI

In other words, we can use Airflow as an orchestrator of the various components and any data transformation operations, and MLflow for serving ML models.

> ***KubeFlow***

> *MLflow and Airflow are very popular frameworks, but there are also other open source serving platforms, Kubeflow being one of them. Kubeflow (https://www.kubeflow.org/) is based on Kubernetes; while MLflow focuses on the lifecycle of the model and Airflow deals with the workflow, Kubeflow is more specific for scalability even if it integrates a workflow and a Jupyter notebook for experimentation.*

In this section, we discovered the serving capabilities of MLflow; in the next section, we will practice with Azure ML. It comes as no surprise that we will once again utilize MLflow in this endeavor.

Working with the Azure ML service

The Azure ML service trains and delivers a model as a containerized application. When we have built the model, we can easily deploy it as a MaaS so that it is very simple to deploy to the Azure cloud. The Azure ML service integrates a user-friendly interface, a computational cluster, experiment tracking and serving with MLflow, and pipeline management.

The basic steps to develop our analytical model with the Azure ML service are as follows:

1. Preparing the data
2. Developing the model with a rich tool, such as Jupyter Notebook, Visual Studio Code, or Azure Notebooks
3. Training and testing the model, using the computational capabilities of Azure ML
4. Registering the model
5. Deploying the model as a microservice

When the model is a web service, we can call from different sources (such as Azure Functions, Azure IoT Hub, or Azure Stream Analytics) and integrate them into our business process. The following diagram demonstrates these concepts:

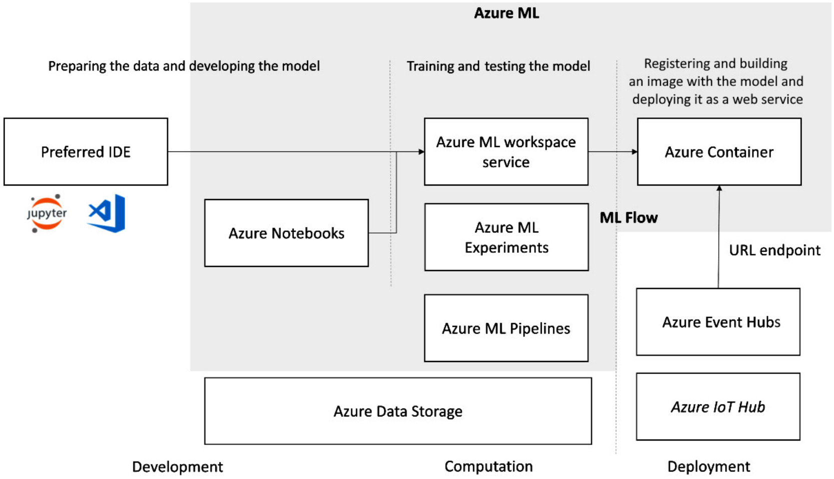

Azure ML
Preparing the data and developing the model
Training and testing the model
Registering and building an image with the model and deploying it as a web service
Preferred IDE
jupyter
Azure Notebooks
Azure ML workspace service
Azure ML Experiments
Azure ML Pipelines
ML Flow
Azure Container
URL endpoint
Azure Event Hubs
Azure IoT Hub
Azure Data Storage
Development
Computation
Deployment

Figure 15.5 – Azure ML development and deploy workflow

We can write our code either using our preferred IDE or directly in a Jupyter notebook exposed by Azure. In either case, we can perform the training and testing phases directly on Azure ML by calling Azure ML APIs. When we are happy with our results, we can expose the model as a web service that can be consumed by other services such as IoT Hub.

Many data scientists prefer to use an integrated graphical environment, such as Azure ML Studio. Others prefer to access a cloud-based environment in which they can write their own code, such as Azure Notebooks. Others prefer to use a local IDE and integrate with the cloud through SDKs so that they can switch between a local environment and a cloud environment using only the APIs. We belong to this last class. In this exercise, we are going to use a locally installed Jupyter notebook with an Azure ML SDK, but we could also use Visual Code with the Azure ML extension (https://visualstudio.microsoft.com/downloads/ai-tools-vscode/). Obviously, it is quite easy to extend the exercise to Azure Notebooks.

Starting with the Azure ML service

To start using the Azure ML service, we can create a workspace by using the Azure portal. From the Azure portal, locate Azure Machine Learning and create a new workspace as shown here:

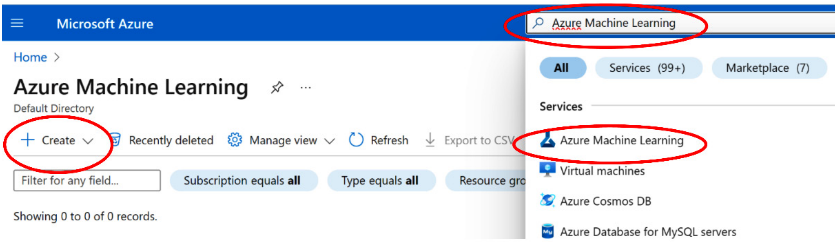
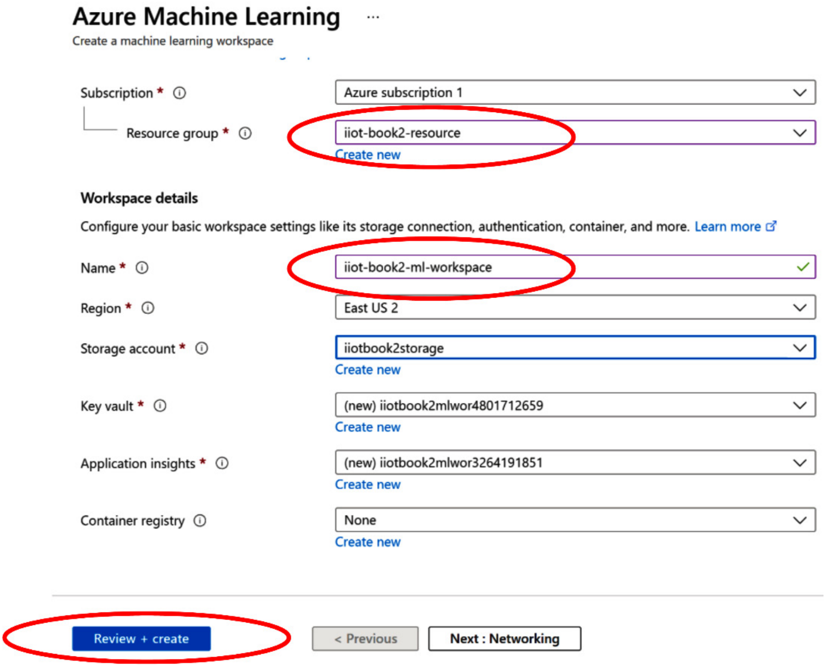

Microsoft Azure
Azure Machine Learning
Home >
Azure Machine Learning
Default Directory
+ Create
Recently deleted
Manage view
Refresh
Export to CSV
Filter for any field...
Subscription equals all
Type equals all
Resource gro
Showing 0 to 0 of 0 records.
All
Services (99+)
Marketplace (7)
Services
Azure Machine Learning
Virtual machines
Azure Cosmos DB
Azure Database for MySQL servers

Figure 15.6 – Creating an Azure ML workspace and service

We then need to provide a name for the workspace, which in our case will be `iiot-book2-ml-workspace`. Choose the standard resource group, `iiot-book2-resource`, and then click on the Review + create button, as shown in the following figure:



Azure Machine Learning ···
Create a machine learning workspace
Create new
Create new
Create new
Create new
Create new
Review + create
< Previous
Next : Networking

Subscription * Azure subscription 1

Resource group * iiot-book2-resource

Workspace details

Configure your basic workspace settings like its storage connection, authentication, container, and more. Learn more

Name * iiot-book2-ml-workspace

Region * East US 2

Storage account * iiotbook2storage

Key vault * (new) iiotbook2mlwor4801712659

Application insights * (new) iiotbook2mlwor3264191851

Container registry None

Figure 15.7 – Azure ML workspace configuration

Understanding the Azure ML workspace

Azure ML facilitates comprehensive ML model development, encompassing data management, EDA, model creation, deployment, and monitoring, as discussed in *Chapter 13* . The primary access point for Azure ML is the workspace (see *Figure 15 .5*).

By engaging with the workspace (through the SDK, CLI, or UI), users can link to their preferred development environment. In the upcoming sections, we will see how to build a wind turbine digital twin working locally with our preferred IDE (Jupyter) or leveraging Azure Notebooks.

Azure ML also allows us to manage datasets, construct modular pipelines, and allocate computing resources (with or without GPU support) for model training and deployment. Azure extensively utilizes Docker containers for both deployment and resource allocation during training.

> ### *MLOps*

> *Azure ML supports end-to-end **continuous integration and continuous delivery** (**CI/CD**) pipelines and retraining pipelines; to understand how to leverage Azure MLOps, see https://learn.microsoft.com/en-us/azure/architecture/ai-ml/guide/machine-learning-operations-v2 .*

We will see in the next section how to deploy our model and build our endpoint to connect it with Azure IoT Hub.

Developing wind turbine digital twins with Azure ML

We developed digital twins for wind turbines in *Chapter 14* . This is a physics-based model and doesn't require ML libraries, which means that it can be deployed on Azure without the Azure ML service. It is, however, a good exercise to understand how Azure ML works. In this exercise, we will follow these steps:

1. Develop a physics-based model with a rich tool such as Jupyter Notebook and Azure Notebooks
2. Develop a more accurate ML model with an Azure notebook
3. Register the model
4. Deploy the model
5. Test the model

Refreshing the model

To avoid replicating the model implementation across multiple sections, we will concisely present the Python code here (**`wind_turbine_model(x)`**), eliminating the need to reiterate it throughout the text:

```
def wind_turbine_model(x):
    # cut-in speed vs cut-out speed
    if x<4.5 or x>21.5:
        return 0.0
    # standard operability
    return 376.936 - 195.8161*x +
33.75734*x**2 - 2.212492*x**3 +
0.06309095*x**4 - 0.0006533647*x**5
reference_power =
[wind_turbine_model(x) for x in
range(0,30)]
```

Now, let’s visualize it:

```
import matplotlib.pyplot as plt
import numpy as np
fig, ax = plt.subplots()
ax.plot(reference_power,'k')
ax.set_xlabel('wind speed (m/s)')
ax.set_ylabel('power (kW)')
plt.show()
```

The output should be similar to the following figure:

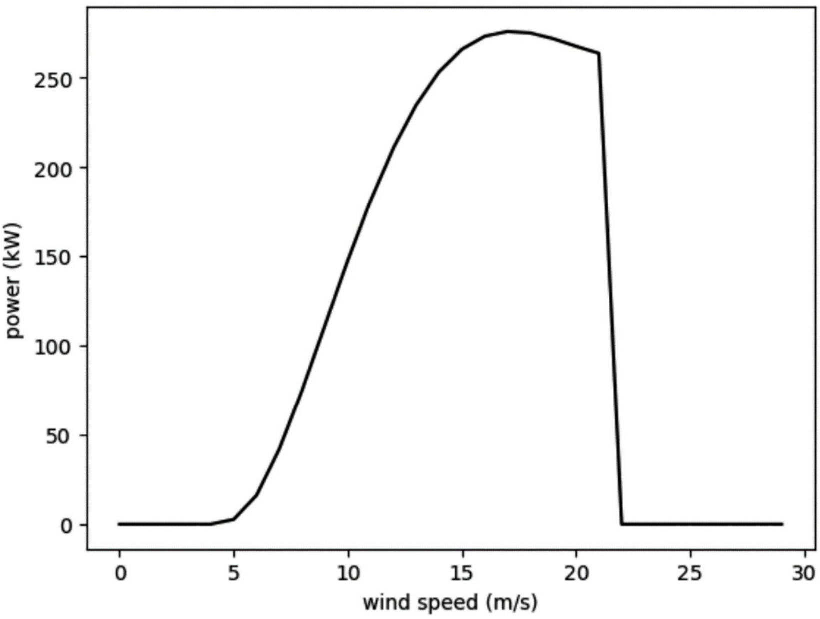


Figure 15.8 – Wind turbine model

Developing the model with Jupyter Notebook

To develop the model, we can either use an IDE or we can use Azure ML Studio Notebooks. In our example, we will use Jupyter Notebook.

We have already become familiar with Jupyter Notebook in *Chapters 13* and *15* , so all we need to do is install Azure ML SDK v2. From the command line, execute the following instructions:

```
$ pip install azureml-core azureml-
sdk[notebooks] jupyter
$ jupyter notebook
```

Alternatively, you can use the Docker image provided with the source code of this book. Access the source code then run the following commands from the Windows or Linux Command Prompt:

```
$ cd Chapter15/AzureML
$ docker compose up -d --build
```

Then, connect to `http://localhost:8888`.

Then, we need to create a new notebook by clicking on New | Python 3 from the menu on the right. As we learned before, we need an instance of the Azure Notebooks workbench so that we can connect to Azure ML. From the workspace, download the `config.json` file and copy it into your working directory (if you are using Docker Compose, copy and paste it into the `work` directory) as shown here:

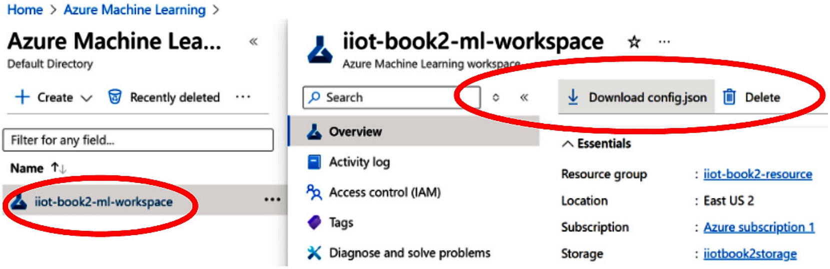

Home > Azure Machine Learning >
Azure Machine Lea...
Default Directory
+ Create
Recently deleted
Filter for any field...
Name
iiot-book2-ml-workspace
iiot-book2-ml-workspace
Azure Machine Learning workspace
Search
Download config.json
Delete
Overview
Activity log
Access control (IAM)
Tags
Diagnose and solve problems
Essentials
Resource group : iiot-book2-resource
Location : East US 2
Subscription : Azure subscription 1
Storage : iiotbook2storage

Figure 15.9 – Downloading config.json from workspace

Now, launch Azure ML Studio, open a new notebook, and execute the following code on the first cell:

```
from azureml.core import Workspace
ws = Workspace.from_config()
```

Connect to the Azure portal as suggested by Azure ML, providing the code of the following output; then, you can see the details. On a new cell, add the following code and execute it:

```
ws.get_details()
```

Now, you are ready to run your experiment. On a new cell, add the following code and execute it:

```
from azureml.core import Experiment
# create a new experiment
exp = Experiment(workspace=ws,
name='wind-turbine-experiment')
# start a run
run = exp.start_logging()
```

Now, log the output of your model. On a new cell, add the following code and execute it:

```
run.log_list('Wind Turbine Model',
reference_power) # log a list of values
run.complete()
```

Finally, get the URL of the experiment. On a new cell, add the following code and execute it:

```
print(run.get_portal_url())
```

Click on the provided link. The output should be similar to the following figure:

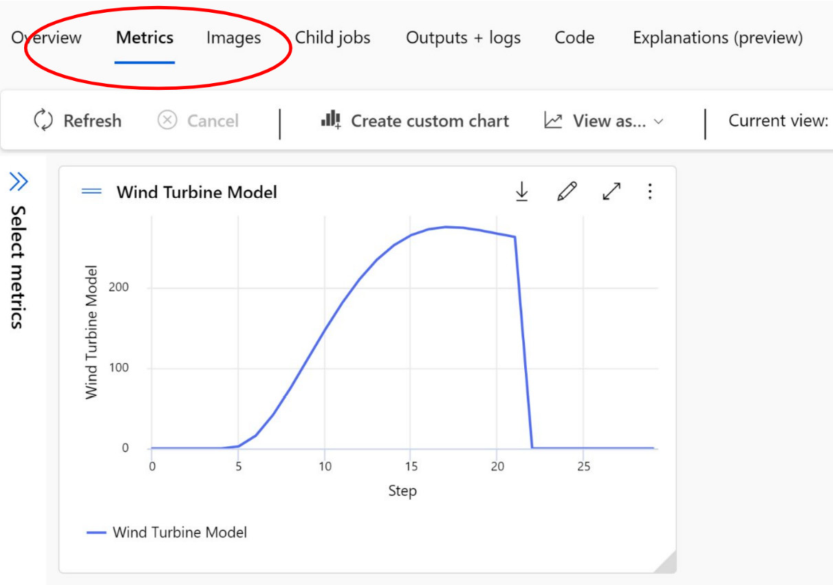

Overview
Metrics
Images
Child jobs
Outputs + logs
Code
Explanations (preview)
Refresh
Cancel
Create custom chart
View as...
Current view:
Select metrics
Wind Turbine Model
Wind Turbine Model
200
100
0
0
5
10
15
20
25
Step
Wind Turbine Model

Figure 15.10 – Tracking experiment on Azure ML

At the end, do not forget to clean up your resources:

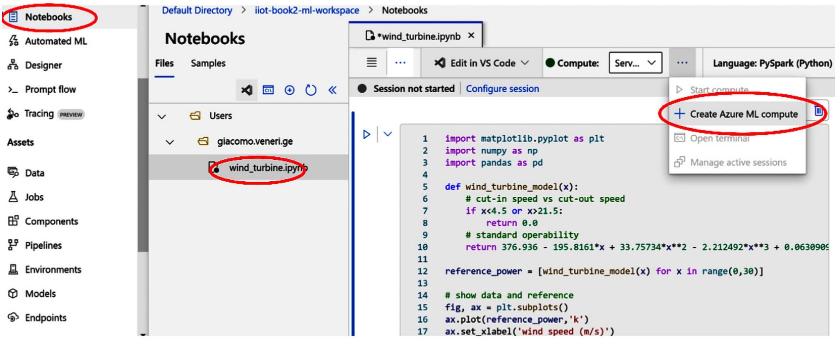
```
# clean the workspace
ws.delete(delete_dependent_resources=True)
```

The model works as expected. We've seen how to work with Azure ML remotely, but we can do the same directly from an Azure ML notebook. Let's see how.

Developing the model with an Azure notebook

The primary benefit of Azure ML is its provision of a robust computing infrastructure tailored for AI and ML. For developing complex models, reliance on a standard PC is insufficient; advanced resources, particularly multiple GPUs, are often necessary.

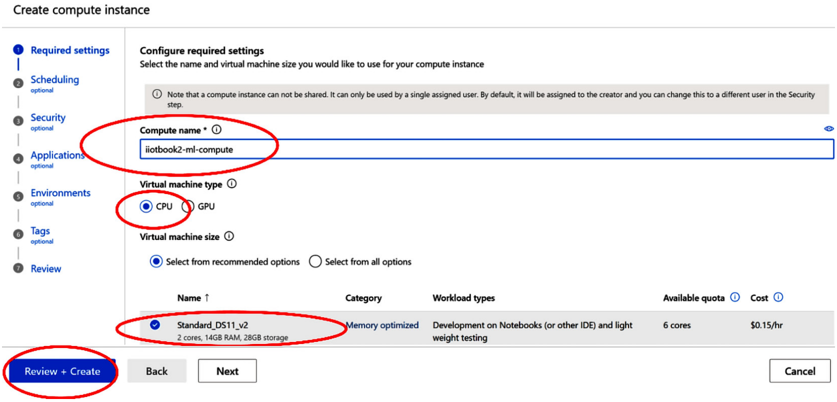
To access computational resources, we can open a new Azure notebook: click on the Notebooks tab in Azure ML Studio, then create a new file, as shown in the following figure. Finally, we need to create a computation ML resource, as shown here:


Notebooks
Automated ML
Designer
Prompt flow
Tracing PREVIEW
Assets
Data
Jobs
Components
Pipelines
Environments
Models
Endpoints
Default Directory > iiot-book2-ml-workspace > Notebooks
Notebooks
Files
Samples
Users
giacomo.veneri.ge
wind_turbine.ipynb
*wind_turbine.ipynb ×
Edit in VS Code
Compute:
Serv...
Language: PySpark (Python)
Session not started
Configure session
Start compute
Create Azure ML compute
Open terminal
Manage active sessions

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

def wind_turbine_model(x):
    # cut-in speed vs cut-out speed
    if x<4.5 or x>21.5:
        return 0.0
    # standard operability
    return 376.936 - 195.8161*x + 33.75734*x**2 - 2.212492*x**3 + 0.063090

reference_power = [wind_turbine_model(x) for x in range(0,30)]

# show data and reference
fig, ax = plt.subplots()
ax.plot(reference_power,'k')
ax.set_xlabel('wind speed (m/s)')
```

Figure 15.11 – Azure notebook

Now, we can set up our computational resources. Provide the compute name, as shown in the following figure. We can avoid GPU support and choose just 6 cores as shown here:



Create compute instance

Required settings

Scheduling
optional

Security
optional

Applications
optional

Environments
optional

Tags
optional

Review

Configure required settings

Select the name and virtual machine size you would like to use for your compute instance

ⓘ Note that a compute instance can not be shared. It can only be used by a single assigned user. By default, it will be assigned to the creator and you can change this to a different user in the Security step.

Compute name * ⓘ

iiotbook2-ml-compute

Virtual machine type ⓘ

● CPU ○ GPU

Virtual machine size ⓘ

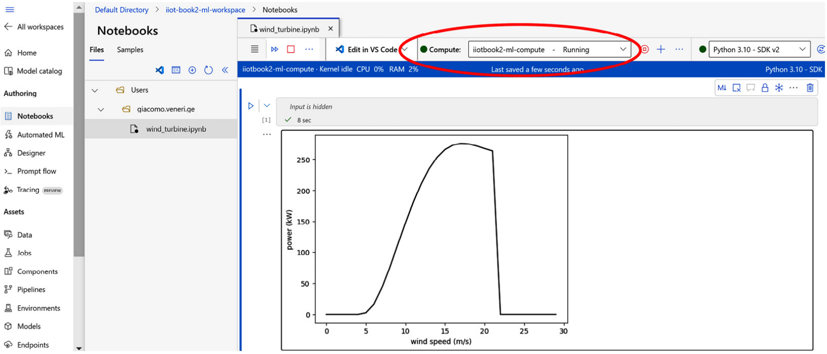
● Select from recommended options ○ Select from all options

	Name ↑	Category	Workload types	Available quota ⓘ	Cost ⓘ
●	Standard_DS11_v2 2 cores, 14GB RAM, 28GB storage	Memory optimized	Development on Notebooks (or other IDE) and light weight testing	6 cores	$0.15/hr

Review + Create
Back
Next
Cancel

Figure 15.12 – Defining compute resource

Now, we can set up our kernel and execute our model. Choose the computing cluster from the drop-down menu as shown ahead, then add a new cell copy and paste the code of our model (see the *Refreshing the model* section), then run the cell. The output should be similar to the following figure:



All workspaces
Home
Model catalog
Authoring
Notebooks
Automated ML
Designer
Prompt flow
Tracing
Assets
Data
Jobs
Components
Pipelines
Environments
Models
Endpoints
Default Directory > iiot-book2-ml-workspace > Notebooks
Notebooks
Files
Samples
Users
giacomo.veneri.ge
wind_turbine.ipynb
Edit in VS Code
Compute: iiotbook2-ml-compute - Running
Python 3.10 - SDK v2
iiotbook2-ml-compute · Kernel idle CPU 0% RAM 2%
Last saved a few seconds ago
Python 3.10 - SDK
Input is hidden
[1]
8 sec
power (kW)
wind speed (m/s)
0
50
100
150
200
250
0
5
10
15
20
25
30

Figure 15.13 – Executing a wind turbine in an Azure notebook

Developing a more accurate ML model with an Azure notebook

As we learned in *Chapter 11* , to develop an analytics model, we can follow five main steps:

1. Defining a problem statement and preparing the data
2. EDA
3. Building the model—either physics-based or data-driven
4. Packaging and deployment
5. Monitoring

In the previous example, we covered these steps in our creation of a physics-based digital-twin model. When we work with a data-driven approach, however, we need an additional sub-step to train and test the model using real data. We can use the ML abilities and high computational abilities (both GPU- and CPU-based) provided by Azure ML to accomplish this goal.

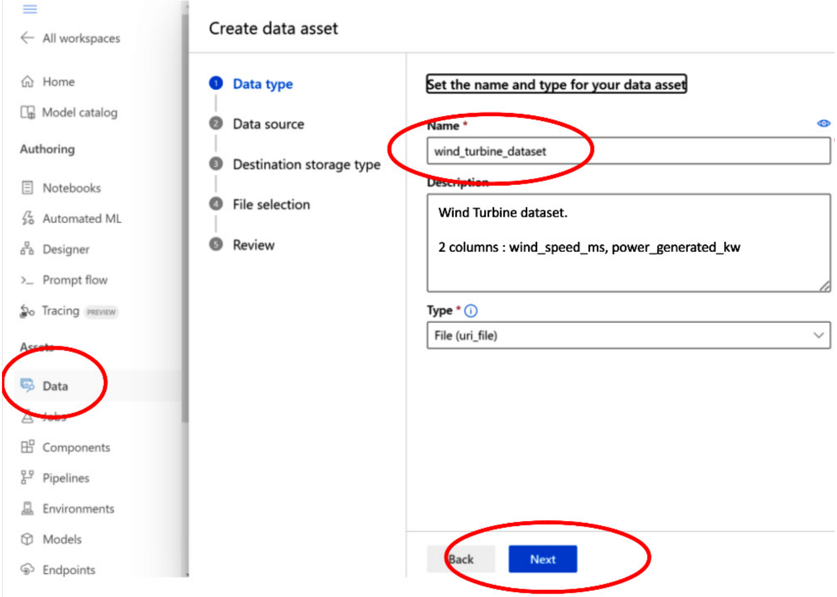
To demonstrate this, we will develop a simple regression model to predict the power generated.

We will use the wind turbine digital twins to produce simulated data to infer a surrogate model from real data. Our exercise is not to find the best algorithm but to use the computational capabilities of the Azure ML service.

Step 1 – Problem statement and dataset

We want to predict power-generated data given the wind speed.

Before starting to develop our model, we need to import our data. From the right menu, select Data | Create data asset with the name `wind_turbine_dataset` as shown here:


All workspaces
Home
Model catalog
Authoring
Notebooks
Automated ML
Designer
Prompt flow
Tracing PREVIEW
Assets
Data
Jobs
Components
Pipelines
Environments
Models
Endpoints
Data type
Data source
Destination storage type
File selection
Review
Back
Next

Create data asset

Set the name and type for your data asset

Name *: wind_turbine_dataset

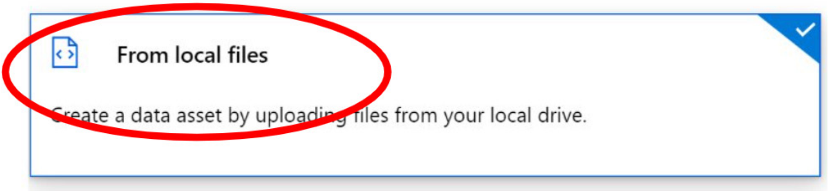
Description: Wind Turbine dataset.

2 columns : wind_speed_ms, power_generated_kw

Type *: File (uri_file)

Figure 15.14 – Creating a new dataset

Import from the local file as shown here:

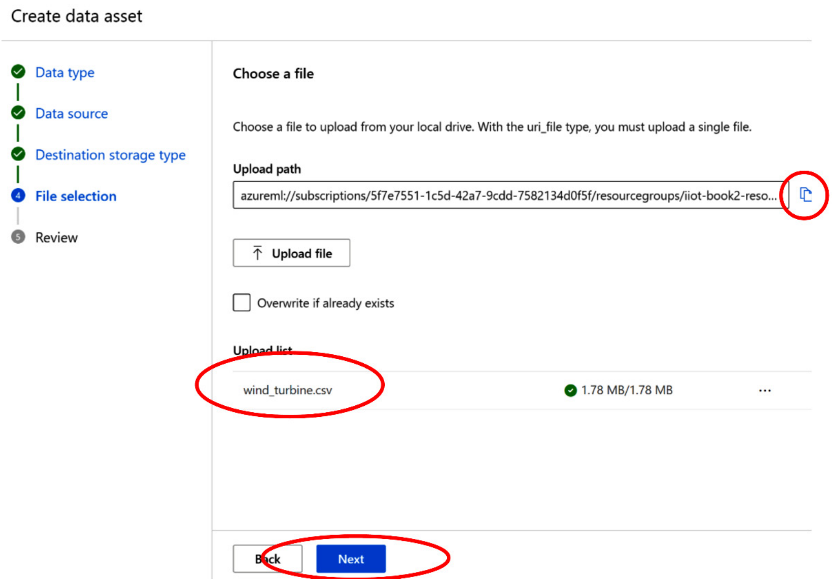


From local files

Create a data asset by uploading files from your local drive.

Figure 15.15 – Importing from local file

Finally, upload the file provided in the official GitHub repository as shown here:


Create data asset

- Data type
- Data source
- Destination storage type
- File selection
- Review

Choose a file

Choose a file to upload from your local drive. With the uri_file type, you must upload a single file.

Upload path

azureml://subscriptions/5f7e7551-1c5d-42a7-9cdd-7582134d0f5f/resourcegroups/iiot-book2-reso...

☐ Overwrite if already exists

Upload list

wind_turbine.csv	1.78 MB/1.78 MB

Figure 15.16 – Uploading data

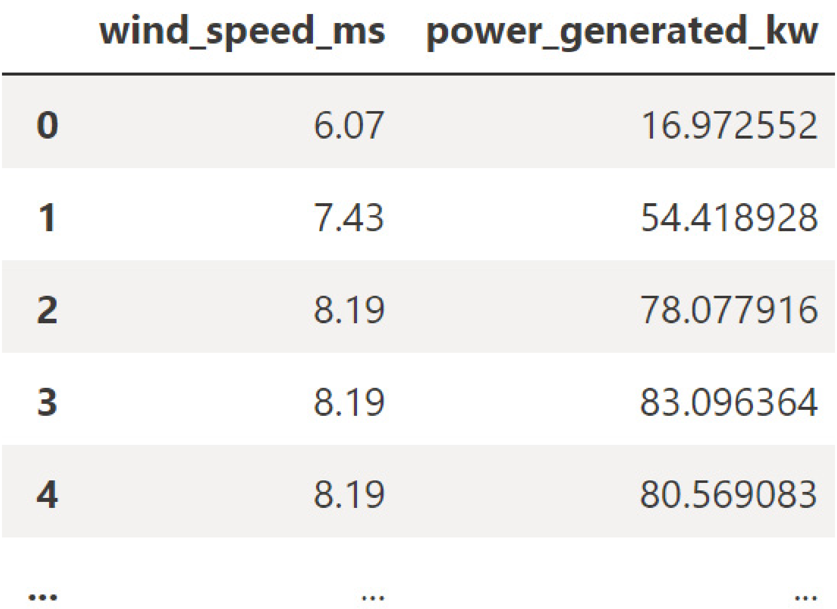
Copy and paste the storage path to use it later.

Step 2 – EDA

Now, we can visualize our data. Let's connect to Azure ML. On a new cell of a notebook, add the following line of code and execute it:

```
import pandas as pd
df = pd.read_csv(<storage path of the
dataset just created>)
df
```

The expected result is something similar to the following figure:



	wind_speed_ms	power_generated_kw
0	6.07	16.972552
1	7.43	54.418928
2	8.19	78.077916
3	8.19	83.096364
4	8.19	80.569083
...	...	...

Figure 15.17 – Wind turbine dataset

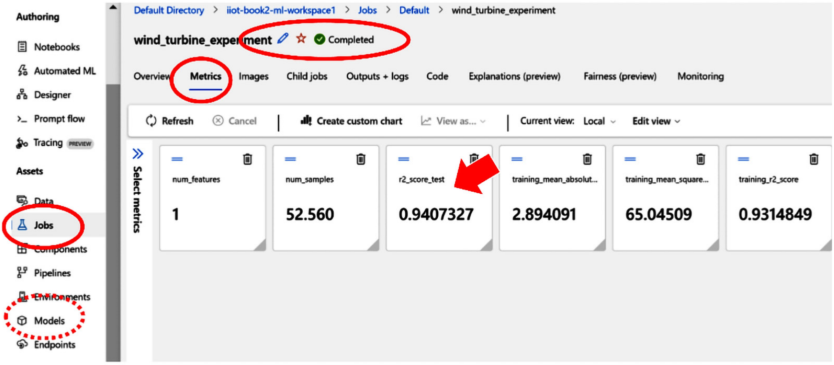
We do not want to perform additional EDA because we have already done it at the beginning of this chapter in the *Developing our ML model* section.

Step 3 – Building the model on an Azure ML notebook

To build the model, we can reuse the code developed in *step 3* of the *Developing our first MaaS with MLflow* section. Create a new file on Azure

Studio | Notebooks and copy and paste the code developed before, then run the cell.

When the job is done, you can see the experiments done from the left menu by selecting Jobs . On the Metrics tab, you can see the metrics logged on our experiment. See the following figure for more details:


Authoring
Notebooks
Automated ML
Designer
Prompt flow
Tracing PREVIEW
Assets
Data
Jobs
Components
Pipelines
Environments
Models
Endpoints
Default Directory > iiot-book2-ml-workspace1 > Jobs > Default > wind_turbine_experiment
wind_turbine_experiment
Completed
Overview
Metrics
Images
Child jobs
Outputs + logs
Code
Explanations (preview)
Fairness (preview)
Monitoring
Refresh
Cancel
Create custom chart
View as...
Current view: Local
Edit view
Select metrics

num_features 1

num_samples 52.560

r2_score_test 0.9407327

training_mean_absolut... 2.894091

training_mean_square... 65.04509

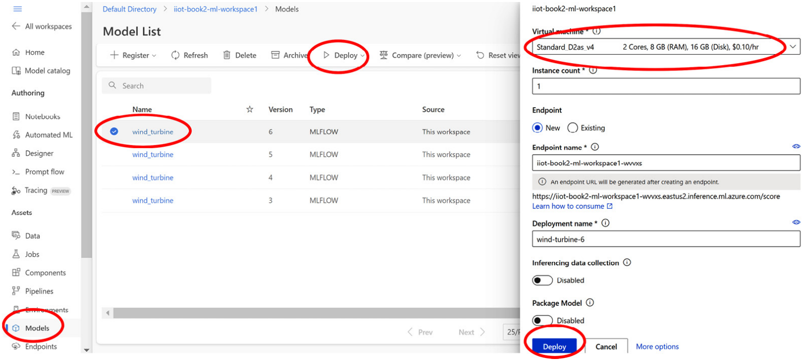
training_r2_score 0.9314849

Figure 15.18 – Metrics tab

As the figure shows, we achieved a 94.0% R2 score, so we can deploy our model.

Step 4 – Packaging and deploying

We have already registered our model. Thus, the next step is deploying the model. From the left menu of Azure ML Studio, click on Models , then select the most recent model and click on Deploy to deploy it as a web service. Finally, define the virtual machine (VM) you want to use and deploy it as shown in the following figure:


All workspaces
Home
Model catalog
Authoring
Notebooks
Automated ML
Designer
Prompt flow
Tracing PREVIEW
Assets
Data
Jobs
Components
Pipelines
Environments
Models
Endpoints
Default Directory > iiot-book2-ml-workspace1 > Models
Model List
Register
Refresh
Delete
Archive
Deploy
Compare (preview)
Reset view
Search
Prev
Next
25/
iiot-book2-ml-workspace1
Learn how to consume
Deploy
Cancel
More options

Name	Version	Type	Source
wind_turbine	6	MLFLOW	This workspace
wind_turbine	5	MLFLOW	This workspace
wind_turbine	4	MLFLOW	This workspace
wind_turbine	3	MLFLOW	This workspace

Virtual machine *: Standard_D2as_v4 2 Cores, 8 GB (RAM), 16 GB (Disk), $0.10/hr

Instance count *: 1

Endpoint: ● New ○ Existing

Endpoint name *: iiot-book2-ml-workspace1-wvvxs

An endpoint URL will be generated after creating an endpoint.

https://iiot-book2-ml-workspace1-wvvxs.eastus2.inference.ml.azure.com/score

Deployment name *: wind-turbine-6

Inferencing data collection: Disabled

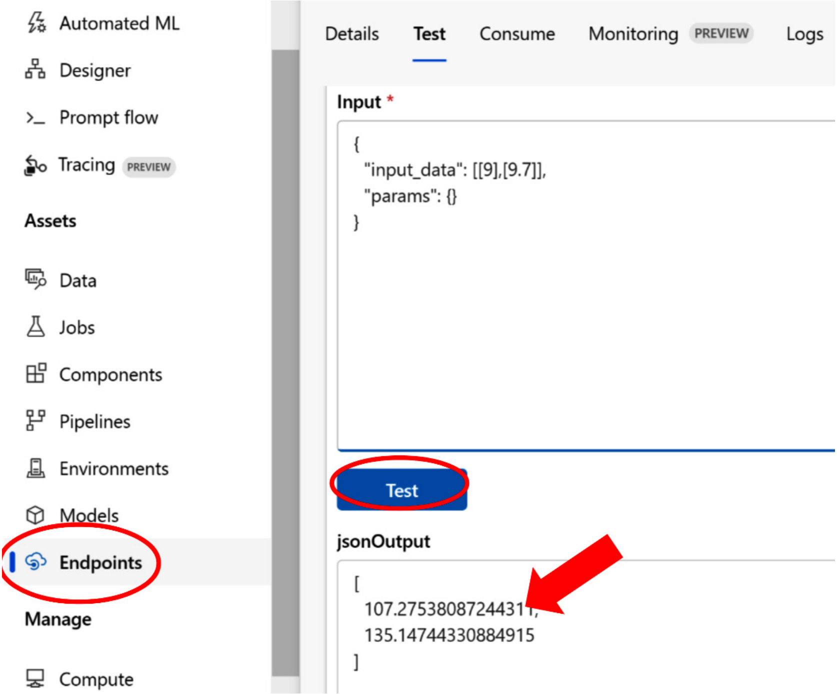
Package Model: Disabled

Figure 15.19 – Deploying the model

It’s time to test our model, so from the left menu, click on the endpoint just created and the Test tab, and then write the following JSON:

```
{
  "input_data": [[9],[9.7]],
  "params": {}
}
```

The output should be similar to the following figure:


Automated ML
Designer
Prompt flow
Tracing PREVIEW
Assets
Data
Jobs
Components
Pipelines
Environments
Models
Endpoints
Manage
Compute
Details
Test
Consume
Monitoring PREVIEW
Logs
Test

Input *

```
{
  "input_data": [[9],[9.7]],
  "params": {}
}
```

jsonOutput

```
[
  107.27538087244311,
  135.14744330884915
]
```

Figure 15.20 – Testing our model

You can also build secure Python code from the Consume tab. Here, you can see the URL and define authentication credentials to use from external services.

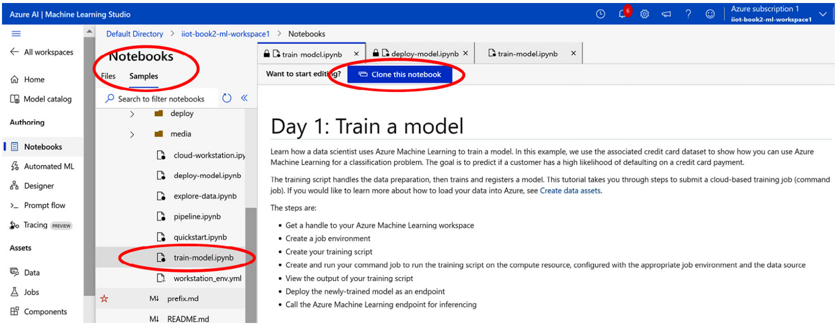
Building the model from a local Jupyter instance

Obviously, you can use your local Jupyter instance to send a command of a training job remotely to

Azure ML. The procedure is a bit more complex:

1. Create the environment with all the dependencies.
2. Create a main file with the training code.
3. Create the training command and send it to the computation cluster.

The easiest way to do this is to use one of the ready-made examples on Azure ML. As shown in the following figure, clone the **`train_model.ipynb`** file found in the samples folder of Azure ML Studio | Notebooks :


Azure AI | Machine Learning Studio
Azure subscription 1
iiot-book2-ml-workspace1
Default Directory
iiot-book2-ml-workspace1
Notebooks
All workspaces
Home
Model catalog
Authoring
Notebooks
Automated ML
Designer
Prompt flow
Tracing PREVIEW
Assets
Data
Jobs
Components
Notebooks
Files
Samples
Search to filter notebooks
deploy
media
cloud-workstation.ipy
deploy-model.ipynb
explore-data.ipynb
pipeline.ipynb
quickstart.ipynb
train-model.ipynb
workstation_env.yml
prefix.md
README.md
train-model.ipynb
deploy-model.ipynb
train-model.ipynb
Want to start editing?
Clone this notebook

Day 1: Train a model

Learn how a data scientist uses Azure Machine Learning to train a model. In this example, we use the associated credit card dataset to show how you can use Azure Machine Learning for a classification problem. The goal is to predict if a customer has a high likelihood of defaulting on a credit card payment.

The training script handles the data preparation, then trains and registers a model. This tutorial takes you through steps to submit a cloud-based training job (command job). If you would like to learn more about how to load your data into Azure, see Create data assets.

The steps are:

- Get a handle to your Azure Machine Learning workspace
- Create a job environment
- Create your training script
- Create and run your command job to run the training script on the compute resource, configured with the appropriate job environment and the data source
- View the output of your training script
- Deploy the newly-trained model as an endpoint
- Call the Azure Machine Learning endpoint for inferencing

Figure 15.21 – Building a notebook to train a model

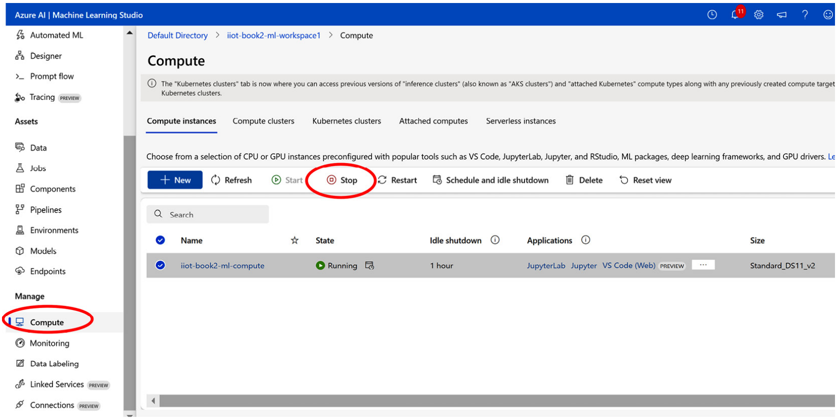
However, we need to make some small changes:

- Provide our access requirements to the ML client as we did before
- Change the model since we will be making a regressor and not a classifier

- Provide our dataset

Cleaning the resources

Please do not forget to clean the resources after your job. As shown in the following figure, access Compute and stop all jobs:


Azure AI | Machine Learning Studio
11
Automated ML
Designer
Prompt flow
Tracing PREVIEW
Assets
Data
Jobs
Components
Pipelines
Environments
Models
Endpoints
Manage
Compute
Monitoring
Data Labeling
Linked Services PREVIEW
Connections PREVIEW
Default Directory > iiot-book2-ml-workspace1 > Compute
Compute
Compute instances
Compute clusters
Kubernetes clusters
Attached computes
Serverless instances
New
Refresh
Start
Stop
Restart
Schedule and idle shutdown
Delete
Reset view
Search

The "Kubernetes clusters" tab is now where you can access previous versions of "inference clusters" (also known as "AKS clusters") and "attached Kubernetes" compute types along with any previously created compute target Kubernetes clusters.

Choose from a selection of CPU or GPU instances preconfigured with popular tools such as VS Code, JupyterLab, Jupyter, and RStudio, ML packages, deep learning frameworks, and GPU drivers. Le

Name	State	Idle shutdown	Applications	Size
iiot-book2-ml-compute	Running	1 hour	JupyterLab Jupyter VS Code (Web) PREVIEW ···	Standard_DS11_v2

Figure 15.22 – Cleaning the resources

Then, remove the endpoint to clean up resources.

IoT Hub integration

The Azure ML service is a way to deliver a web application that can be consumed through a REST

API by other services. In *Chapter 12* , we learned how to connect the IoT Hub service to Stream Analytics and define a JavaScript custom function. We can follow the same procedure here by adding an Azure ML function from the Functions tab and providing the URL. These steps are shown in the following figure:

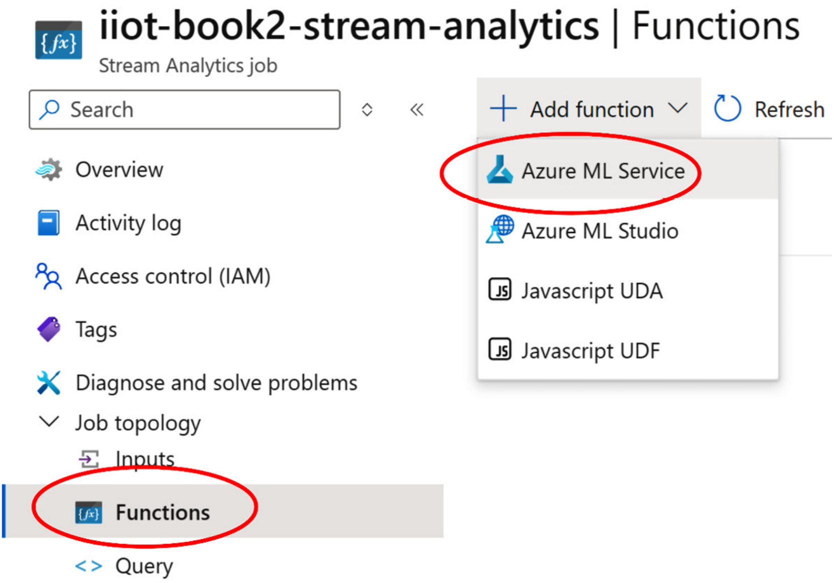
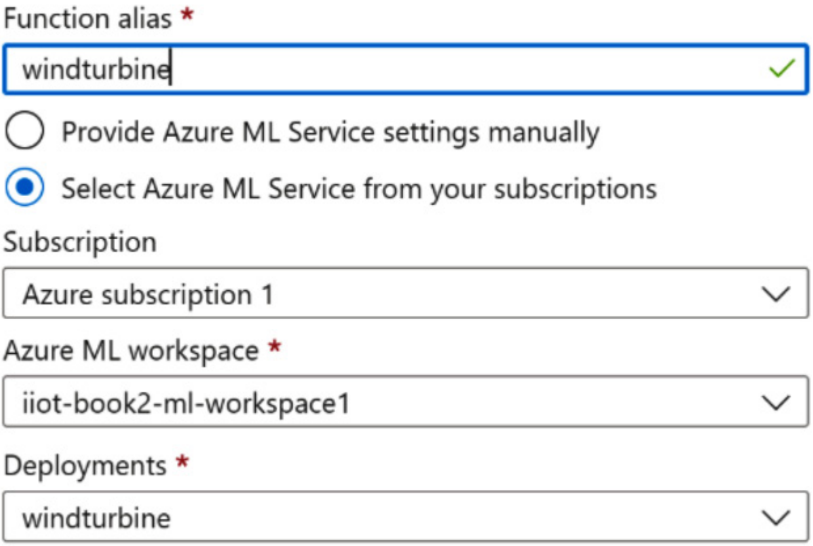

iiot-book2-stream-analytics | Functions
Stream Analytics job
Search
Add function
Refresh
Azure ML Service
Azure ML Studio
Javascript UDA
Javascript UDF
Overview
Activity log
Access control (IAM)
Tags
Diagnose and solve problems
Job topology
Inputs
Functions
Query

Figure 15.23 – Connecting IoT Hub and Azure ML

Then, use the just-created web service as shown here:


Function alias *

windturbine

○ Provide Azure ML Service settings manually

● Select Azure ML Service from your subscriptions

Subscription

Azure subscription 1

Azure ML workspace *

iiot-book2-ml-workspace1

Deployments *

windturbine

Figure 15.24 – Selecting the windturbine web service

Finally, you can use your function – for instance, like this:

```
SELECT udf.windturbine(9) INTO output
FROM input
```

Azure IoT Edge

In some cases, it is necessary to deploy and run ML analytics directly on the edge. Such cases include the following:

- Offline model : When frequent disconnections necessitate models to operate independently without relying on a constant internet connection
- Low latency : When the model needs to respond quickly – for example, when it needs to interact with a controller
- Cost efficiency : When one wants to save costs on the cloud and increase scalability
- Privacy, compliance, and regulation : If it is not possible to send all the data to the cloud due to data export limits or privacy concerns (for example, if we process images with people)
- Bandwidth : If we need to operate with massive data such as high-frequency vibrations and we do not have enough bandwidth

In *Chapter 11* , we learned that Azure IoT Edge supports module integration via Docker. Azure suggests deploying the ML models to the cloud via Azure ML, then saving them to a shared blob and downloading them using the power of IoT Hub's module twins.

> ## *Saving models*

> *There are many formats for saving models.* ***Pickle*** *is the format supported by Python for saving objects (Pickle is considered unsecured, please be careful*

> *when leveraging this format). **Joblib** is a light alternative to Pickle (https://joblib.readthedocs.io). **H5** is supported by TensorFlow for saving weights (https://www.tensorflow.org/guide/keras/serialization_and_saving). PyTorch supports a native format (https://pytorch.org/tutorials/beginner/saving_loading_models.html). **Open Neural Network eXchange** (**ONNX**) is an open-standard format designed for representing ML models, backed by a community of partners that have integrated it into various frameworks and tools (https://onnx.ai/).*

We already saw how to deploy edge-side analytics in *Chapter 11* ; for more information, see https://learn.microsoft.com/en-us/azure/architecture/guide/iot/machine-learning-inference-iot-edge .

In this section, we developed our model leveraging Azure ML Studio. In the next section, we will do the same exercise using AWS.

Implementing analytics on Amazon SageMaker

Amazon SageMaker is a fully managed service that enables data scientists to build, train, and deploy ML models at any scale. Amazon SageMaker is based on Jupyter Notebook so that developers can use a familiar user interface to build their own analytics. The basic concepts of SageMaker are the same as Azure ML. We can build our analytics on Jupyter and our training cluster through a Python API, and then deploy our MaaS that can be consumed through a REST API. SageMaker also supports foundational models to develop models on already existing pre-trained models.

SageMaker Studio is a full platform able to manage the entire model lifecycle development: from data management, model development, training or fine-tuning, experiment tracking, and delivering.

The basic steps to develop our analytical model with the Azure ML service are as follows:

1. Preparing the data
2. Developing the model with a rich tool, such as a Jupyter Notebook
3. Training and testing the model, using the computational capabilities of Azure ML
4. Deploying the MaaS

When the model is a web service, we can call it from different sources (such as AWS IoT Analytics) and integrate them into our business process.

Accessing SageMaker Studio

Before starting our exercise, we need to start SageMaker Studio. From the AWS portal, search for `SageMaker` and click on it as shown in the following figure:

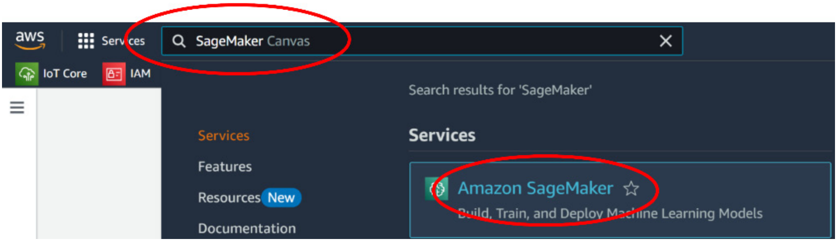
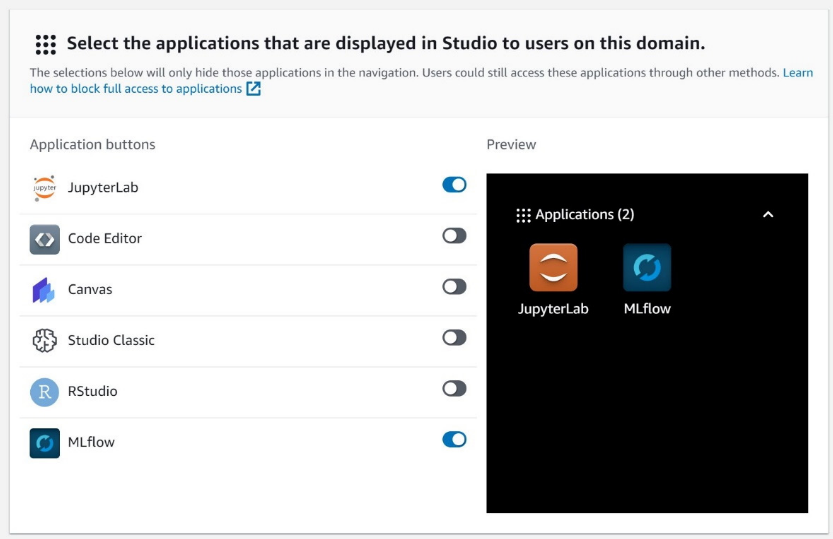

aws
Services
SageMaker Canvas
IoT Core
IAM
Search results for 'SageMaker'
Services
Features
Resources New
Documentation
Services
Amazon SageMaker
Build, Train, and Deploy Machine Learning Models

Figure 15.25 – Accessing SageMaker

SageMaker requires you to set up a domain; for our exercise, we can set up SageMaker for a single user . In the next screen, provide a valid name for the studio and enable only the required applications, as shown in the following figure:



Select the applications that are displayed in Studio to users on this domain.

The selections below will only hide those applications in the navigation. Users could still access these applications through other methods. Learn how to block full access to applications

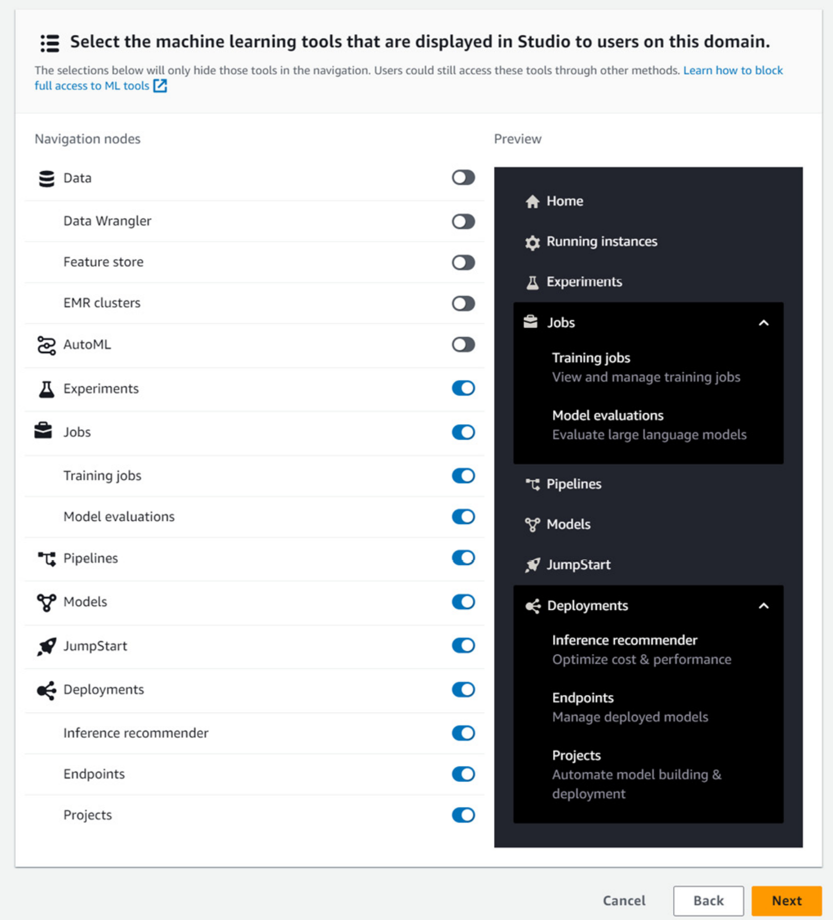
Application buttons

- JupyterLab ●
- Code Editor ○
- Canvas ○
- Studio Classic ○
- RStudio ○
- MLflow ●

Preview

Figure 15.26 – Required applications

For our experiment, we do not need advanced data management and AutoML, as shown in the following figure:



Select the machine learning tools that are displayed in Studio to users on this domain.

The selections below will only hide those tools in the navigation. Users could still access these tools through other methods. Learn how to block full access to ML tools

Navigation nodes

- Data ○
  - Data Wrangler ○
  - Feature store ○
  - EMR clusters ○
- AutoML ○
- Experiments ●
- Jobs ●
  - Training jobs ●
  - Model evaluations ●
- Pipelines ●
- Models ●
- JumpStart ●
- Deployments ●
  - Inference recommender ●
  - Endpoints ●
  - Projects ●

Preview
Home
Running instances
Experiments
Jobs
Training jobs
View and manage training jobs
Model evaluations
Evaluate large language models
Pipelines
Models
JumpStart
Deployments
Inference recommender
Optimize cost & performance
Endpoints
Manage deployed models
Projects
Automate model building & deployment
Cancel
Back
Next

Figure 15.27 – Required ML tools

Finally, review, and from the left menu, click on Studio | Open Studio .

Preparing an S3 bucket to store data

When developing an AI model, it is good practice to organize the data in a single repository. Before proceeding, we suggest creating an S3 bucket where you can upload the data and grant SageMaker access to that S3 bucket.

To create an S3 bucket from the AWS S3 portal, search for `s3` as shown in the following figure:

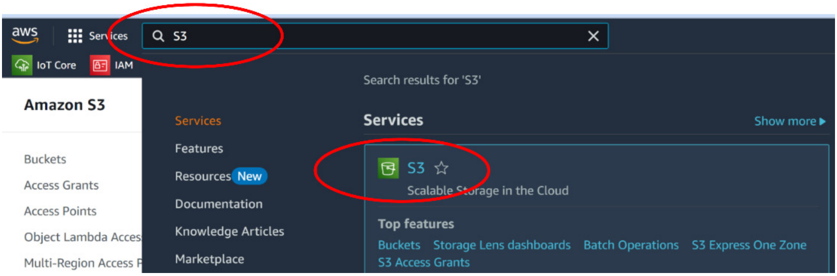
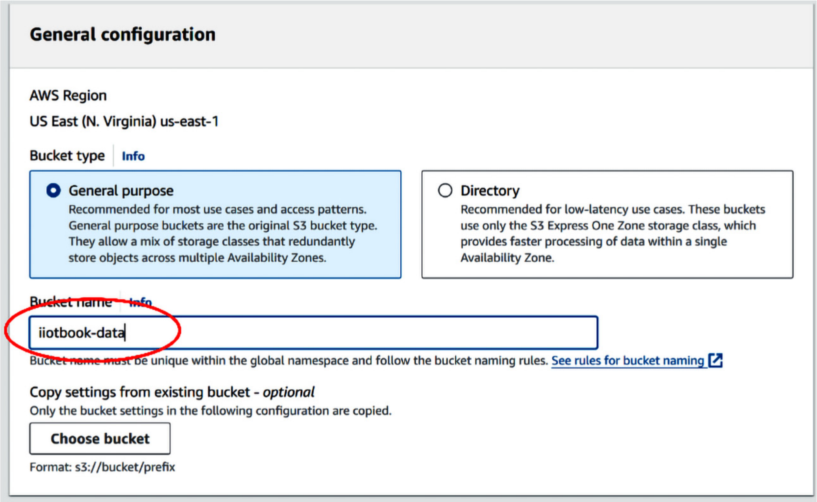

aws
Services
S3
IoT Core
IAM
Amazon S3
Buckets
Access Grants
Access Points
Object Lambda Acces
Multi-Region Access P
Search results for 'S3'
Services
Features
Resources New
Documentation
Knowledge Articles
Marketplace
Services
Show more ▶
S3 ☆
Scalable Storage in the Cloud
Top features
Buckets
Storage Lens dashboards
Batch Operations
S3 Express One Zone
S3 Access Grants

Figure 15.28 – Locating an S3 bucket on the AWS portal

Create an S3 bucket named `iiotbook-data`, as shown in the following figure:



General configuration

AWS Region

US East (N. Virginia) us-east-1

Bucket type | Info

● General purpose
Recommended for most use cases and access patterns. General purpose buckets are the original S3 bucket type. They allow a mix of storage classes that redundantly store objects across multiple Availability Zones.

○ Directory
Recommended for low-latency use cases. These buckets use only the S3 Express One Zone storage class, which provides faster processing of data within a single Availability Zone.

Bucket name | Info

iiotbook-data

Bucket name must be unique within the global namespace and follow the bucket naming rules. See rules for bucket naming

Copy settings from existing bucket - *optional*

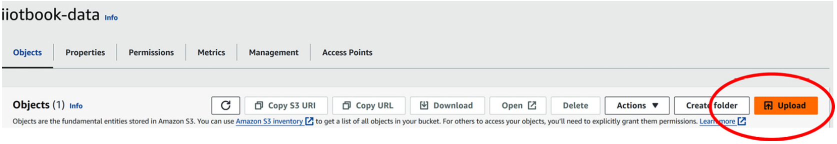
Only the bucket settings in the following configuration are copied.

Choose bucket

Format: s3://bucket/prefix

Figure 15.29 – Building a new bucket

Then, upload the **`windturbine.csv`** file, which you can find in the official repository of the book. From the bucket, just click Create , locate the Objects tab, and click on Upload , as shown in the following figure:



iiotbook-data Info

Objects
Properties
Permissions
Metrics
Management
Access Points

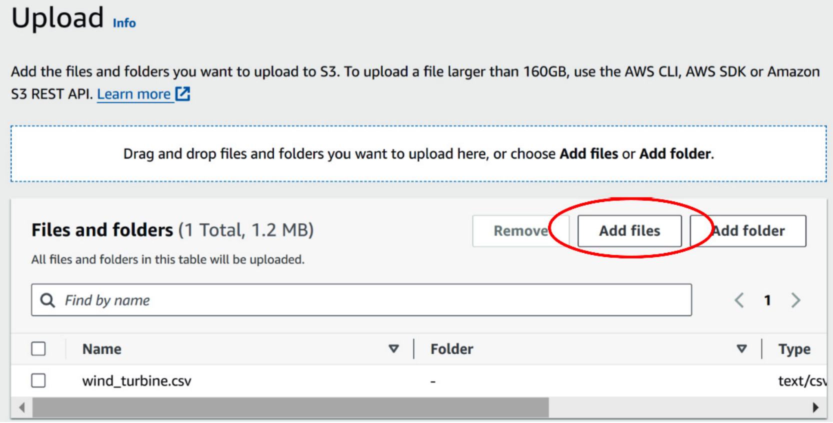
Objects (1) Info

Copy S3 URI
Copy URL
Download
Open
Delete
Actions ▼
Create folder
Upload

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Figure 15.30 – Starting uploading

Now, you can upload the file, as shown in the following figure:



Upload Info

Add the files and folders you want to upload to S3. To upload a file larger than 160GB, use the AWS CLI, AWS SDK or Amazon S3 REST API. Learn more

Drag and drop files and folders you want to upload here, or choose **Add files** or **Add folder**.

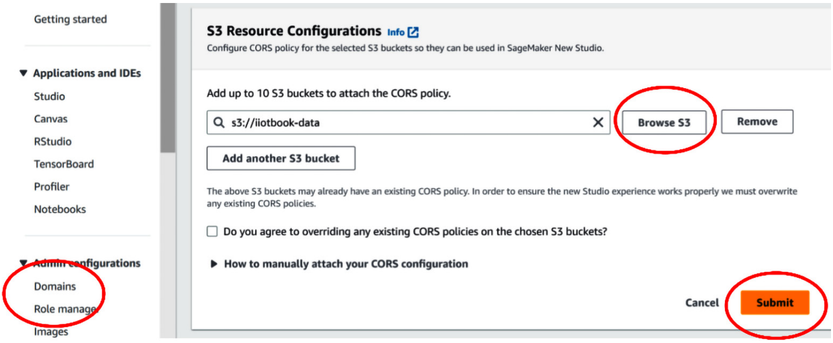
Files and folders (1 Total, 1.2 MB)
Remove
Add files
Add folder
Find by name
1
Name
Folder
Type

All files and folders in this table will be uploaded.

☐	Name	Folder	Type
☐	wind_turbine.csv	-	text/csv

Figure 15.31 – Uploading data to S3 bucket

Finally, from the left menu of SageMaker, locate the domain just created and enable access to the S3 bucket, as shown in the following figure:



Getting started
Applications and IDEs
Studio
Canvas
RStudio
TensorBoard
Profiler
Notebooks
Admin configurations
Domains
Role manage
Images
S3 Resource Configurations Info
Browse S3
Remove
Add another S3 bucket
How to manually attach your CORS configuration
Cancel
Submit

Configure CORS policy for the selected S3 buckets so they can be used in SageMaker New Studio.

Add up to 10 S3 buckets to attach the CORS policy.

s3://iiotbook-data

The above S3 buckets may already have an existing CORS policy. In order to ensure the new Studio experience works properly we must overwrite any existing CORS policies.

☐ Do you agree to overriding any existing CORS policies on the chosen S3 buckets?

Figure 15.32 – Granting access to the S3 bucket to SageMaker

Now, we’ll start development.

Developing a digital twin with SageMaker

We aim to replicate the previous exercise conducted with Azure ML to create a digital twin of a wind turbine, but this time, we will utilize SageMaker.

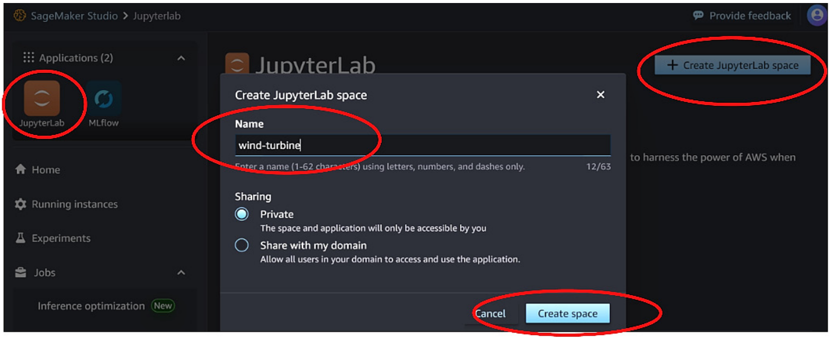
The steps we need to follow to build the analytics are listed as follows:

- Download the dataset from the Amazon S3 bucket.
- Create a new notebook instance in SageMaker Studio.
- Explore the dataset through a notebook and create training data and a test dataset.
- Train a model using the SageMaker distributed computing architecture.
- Host the model in the SageMaker-managed hosting infrastructure and expose an inference endpoint.
- Test the endpoint and connect the model's endpoint to the data source.

Step 1 – Problem statement, environment, and dataset

The problem statement is to build a surrogate model of a wind turbine given a real dataset.

Let’s start our development environment. Start a new Jupyter notebook instance as shown in the following figure. Go to JupyterLab | Create JupyterLab space and provide a name:



SageMaker Studio > Jupyterlab
Provide feedback
Applications (2)
JupyterLab
MLflow
Home
Running instances
Experiments
Jobs
Inference optimization
New
JupyterLab
Create JupyterLab space
to harness the power of AWS when
Cancel
Create space

Create JupyterLab space

Name

wind-turbine

Enter a name (1-62 characters) using letters, numbers, and dashes only. 12/63

Sharing

● Private

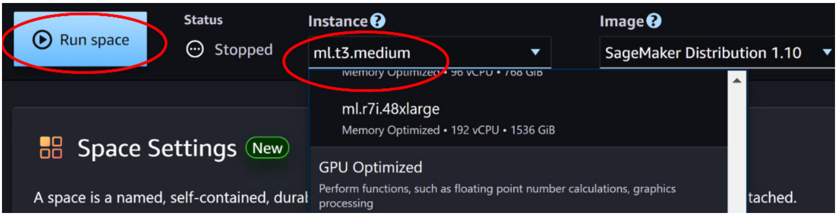
The space and application will only be accessible by you

○ Share with my domain

Allow all users in your domain to access and use the application.

Figure 15.33 – Starting JupyterLab notebook on SageMaker

Then, SageMaker will ask you the instance where to run the training. For this exercise, we suggest using the `t3.medium` size, but you can also leverage GPU support, as shown in the following figure:


Run space
Space Settings
New
A space is a named, self-contained, durab
Memory Optimized • 96 vCPU • 768 GiB
ml.r7i.48xlarge
Memory Optimized • 192 vCPU • 1536 GiB
GPU Optimized
Perform functions, such as floating point number calculations, graphics processing
tached.

Status: Stopped

Instance: ml.t3.medium

Image: SageMaker Distribution 1.10

Figure 15.34 – Choosing the instance where to run Jupyter

Finally, click on Open JupyterLab , and you will be redirected to something we already know: the Jupyter environment. Open a new Jupyter notebook and install the `s3fs` package to access the S3 bucket. On a new cell, copy and execute the following command:

```
%pip install s3fs
```

> ***Jupyter pip install and magic commands***

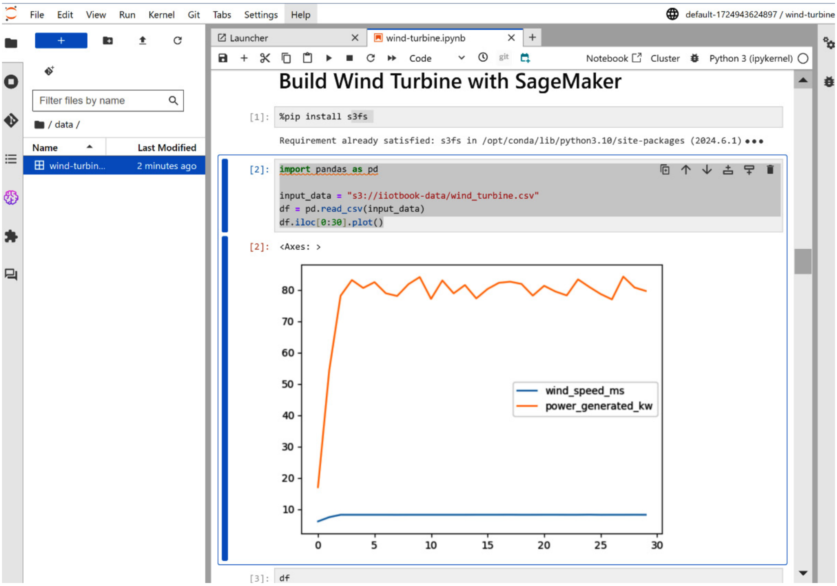
> *Jupyter allows you to perform installation and Bash execution using the* `%` *and* `!` *magic commands. Do not forget to restart the kernel after installation.*

Step 2 – EDA

Now, we can visualize our data. Let's connect to Azure ML. On a new cell of the notebook, add the following line of code and execute it:

```
import pandas as pd
input_data = "s3://iiotbook-data/
wind_turbine.csv"
df = pd.read_csv(input_data)
df.iloc[0:30].plot()
```

The expected output is shown in the following figure:


File
Edit
View
Run
Kernel
Git
Tabs
Settings
Help
default-1724943624897 / wind-turbine
Filter files by name
/ data /
Name
Last Modified
wind-turbin...
2 minutes ago
Launcher
wind-turbine.ipynb
Code
Notebook
Cluster
Python 3 (ipykernel)
80
70
60
50
40
30
20
10
0
5
10
15
20
25
30
wind_speed_ms
power_generated_kw

Build Wind Turbine with SageMaker

```
[1]: %pip install s3fs

Requirement already satisfied: s3fs in /opt/conda/lib/python3.10/site-packages (2024.6.1) •••

[2]: import pandas as pd

     input_data = "s3://iiotbook-data/wind_turbine.csv"
     df = pd.read_csv(input_data)
     df.iloc[0:30].plot()

[2]: <Axes: >

[3]: df
```

Figure 15.35 – Showing data on SageMaker

Step 3 – Building the model using SageMaker

To build the model, we can reuse the code developed in *step 3* of the *Developing our first MaaS with MLflow* section. From SageMaker, go to Studio | Notebooks and copy and paste the following code, then run the cell:

```
import pandas as pd
from sklearn.ensemble import
GradientBoostingRegressor
from sklearn.metrics import r2_score
from sklearn.model_selection import
train_test_split
train_df, test_df = train_test_split(
    df,
    test_size=0.3,
)
y_train =
train_df.pop("power_generated_kw")
X_train = train_df.values
y_test =
test_df.pop("power_generated_kw")
X_test = test_df.values
reg = GradientBoostingRegressor(
    n_estimators=n_estimators
)
```

```
reg.fit(X_train, y_train)
y_pred = reg.predict(X_test)
print(r2_score(y_test, y_pred))
```

The expected output is about `0.9324`. The model's performance is good, so we can fully train the model on the full dataset and deploy it. We can leverage **`sagemaker.sklearn.estimator.SKLearn`** to achieve this goal.

On a new cell, add the following code to send data to the SageMaker session:

```
PREFIX = "wind-turbine"
WORK_DIRECTORY = "data"
import sagemaker,os
sagemaker_session =
sagemaker.Session()
os.makedirs(WORK_DIRECTORY,
exist_ok=True)
df.to_csv("{}/
{}.csv".format(WORK_DIRECTORY,
PREFIX), index=False)
train_input =
sagemaker_session.upload_data(
    WORK_DIRECTORY, key_prefix="{}/
{}".format(PREFIX, WORK_DIRECTORY)
```

```
    )
```

Now, we can submit a training job, but before that, we need to define the training script. The training script is a Python file with the following methods:

- **`__main__`** : This method is called for training
- **`model_fn(model_dir)`** : This method is for loading the model
- **`predict_fn(input_data, model)`** : This method is called to execute a prediction

SageMaker also customizes serialization and deserialization of the request, but in our exercise, we will use the default method:

- **`output_fn(prediction, response_content_type)`** : This method is used to serialize the prediction
- **`input_fn(request_body, request_content_type`** : This method is used to deserialize the request

Let's implement our training method. On a new cell, add the following code:

```
%%writefile wind_turbine.py
from __future__ import print_function
import argparse, joblib, os
import pandas as pd
import numpy as np
from sklearn.ensemble import
GradientBoostingRegressor
from sklearn.metrics import r2_score
if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    # Hyperparameters are described
here. In this simple example we are
just including one hyperparameter.
    parser.add_argument('--n-
estimators', type=int, default=-1)
    # Sagemaker specific arguments.
Defaults are set in the environment
variables.
    parser.add_argument('--output-
data-dir', type=str,
default=os.environ['SM_OUTPUT_DATA_DIR'])
    parser.add_argument('--model-dir',
type=str,
default=os.environ['SM_MODEL_DIR'])
    parser.add_argument('--train',
type=str,
default=os.environ['SM_CHANNEL_TRAIN'])
    args = parser.parse_args()
    input_files = [
os.path.join(args.train, file) for
file in os.listdir(args.train) if
file.endswith(".csv") ]
        raw_data = [ pd.read_csv(file,
engine="python") for file in
input_files ]
    train_df = pd.concat(raw_data)
    ... # code for training
```

Now, we can add the previous code to train the model. Finally, save the model using `joblib`. After the training code, save the model using the following line:

```
    joblib.dump(reg,
os.path.join(args.model_dir,
"model.joblib"))
```

On the same cell, also implement the function to load the model and make the prediction, as shown here:

```
def model_fn(model_dir):
    return
joblib.load(os.path.join(model_dir,
"model.joblib"))
```

Then, add also the following code to make a prediction:

```
def predict_fn(input_data, model):
    if len(input_data.shape) <2:
        input_data =
input_data.reshape(-1,1)
    prediction =
model.predict(input_data)
    return np.array(prediction)
```

Now, we can submit the job using the following code. **`scikit-learn`** does not use GPUs, so we can deploy it on a CPU-only instance (**`ml.c4.xlarge`**), but the main advantage of SageMaker is to optimize processes that take a long time (from a few hours to weeks). On a new cell, add and run the following code:

```
from sagemaker.sklearn.estimator
import SKLearn
from sagemaker import
get_execution_role
sklearn = SKLearn(
    entry_point= "wind_turbine.py",
    framework_version= "1.2-1",
    instance_type="ml.c4.xlarge",
    role= get_execution_role(),
    sagemaker_session=sagemaker_session,
    hyperparameters={"n-estimators":
100},
)
```

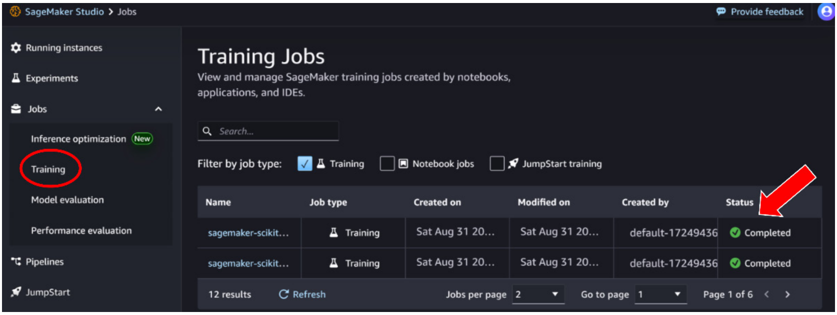
> ### *Additional packages to install*

> *If you need additional packages, you can write your* `requirements.txt` *file and add it as additional source code on the* `Sklearn` *processor; see the MLflow section for an example.*

SageMaker will send the previously created Python code to the training job and will execute it with the following code:

```
sklearn.fit({"train": train_input})
```

We can check the job's stats from the SageMaker UI. From the left menu, go to Jobs | Training , as shown in the following figure:


SageMaker Studio > Jobs
Provide feedback
Running instances
Experiments
Jobs
Inference optimization
New
Training
Model evaluation
Performance evaluation
Pipelines
JumpStart
Search...
Refresh

Training Jobs

View and manage SageMaker training jobs created by notebooks, applications, and IDEs.

Filter by job type: ☑ Training ☐ Notebook jobs ☐ JumpStart training

Name	Job type	Created on	Modified on	Created by	Status
sagemaker-scikit...	Training	Sat Aug 31 20...	Sat Aug 31 20...	default-17249436	✓ Completed
sagemaker-scikit...	Training	Sat Aug 31 20...	Sat Aug 31 20...	default-17249436	✓ Completed

12 results

Jobs per page 2

Go to page 1

Page 1 of 6

Figure 15.36 – Checking training job

If we are happy, we can deploy our model.

Step 4 – Packaging and deploying

We have already registered our model. Thus, the next step is deploying the model. On a new cell, add the following line of code:

```
predictor =
sklearn.deploy(endpoint_name="windturbine",
initial_instance_count=1,
instance_type="ml.m5.xlarge")
```

Finally, we can test it. On a new cell, add the following code and execute it:

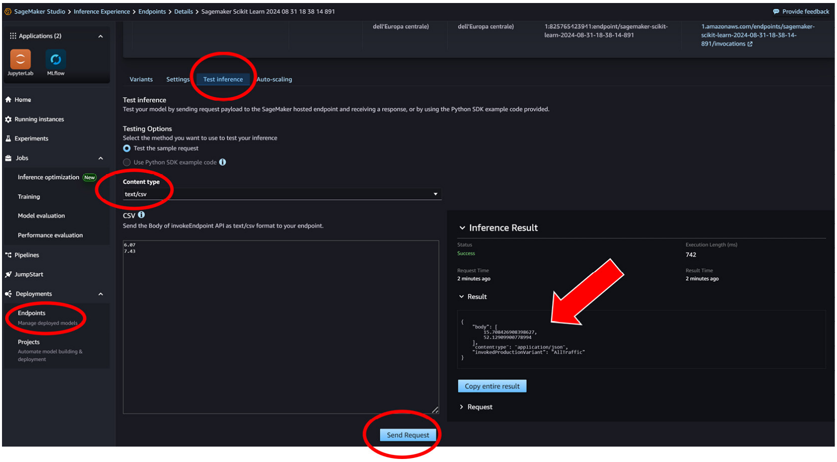
```
predictor.predict(X_test)
```

The expected result is an array similar to the following:

```
array([89.59522204, 67.88331554,
72.57244909, ..., 72.57244909,
       28.46291311, 43.59427434])
```

You can also test the model from the UI, `wget` , or Postman. From the left menu, go to SageMaker | Endpoints , then click on Test inference . As shown in the following figure, choose text/csv and send the following data:

```
  6.07
7.43
```


SageMaker Studio > Inference Experience > Endpoints > Details > Sagemaker Scikit Learn 2024 08 31 18 38 14 891
Provide feedback
Applications (2)
JupyterLab
MLflow
Home
Running instances
Experiments
Jobs
Inference optimization New
Training
Model evaluation
Performance evaluation
Pipelines
JumpStart
Deployments
Endpoints
Manage deployed models
Projects
Automate model building & deployment
Variants
Settings
Test inference
Auto-scaling
Copy entire result
Request
Send Request

dell'Europa centrale) | dell'Europa centrale) | 1:825765423941:endpoint/sagemaker-scikit-learn-2024-08-31-18-38-14-891 | 1.amazonaws.com/endpoints/sagemaker-scikit-learn-2024-08-31-18-38-14-891/invocations

Test inference

Test your model by sending request payload to the SageMaker hosted endpoint and receiving a response, or by using the Python SDK example code provided.

Testing Options

Select the method you want to use to test your inference

● Test the sample request

○ Use Python SDK example code

Content type: text/csv

CSV

Send the Body of invokeEndpoint API as text/csv format to your endpoint.

```
6.07
7.43
```

Inference Result

Status: Success

Execution Length (ms): 742

Request Time: 2 minutes ago

Result Time: 2 minutes ago

Result

```
{
    "body": [
        15.708426908398627,
        52.12909900778994
    ],
    "contentType": "application/json",
    "invokedProductionVariant": "AllTraffic"
}
```

Figure 15.37 – Testing our endpoint

> ## ***Logging info***

> *Error and actions of the deployed model are tracked on Amazon CloudWatch. You can access it to see errors and logs.*

The model is ready, so we can now consume it from AWS IoT Core.

Consuming the model from AWS IoT Core

When the model is ready, how do you connect IoT data to the model? In *Chapters 10* and *12* , we proposed our architecture be based on cold and hot paths. If you want to develop a cold path, the only thing to do is to leverage a scheduled service (such as Amazon EventBridge) and connect a time series database or S3 bucket to the SageMaker model.

As we saw in *Chapter 10* , we can leverage AWS IoT Analytics to connect AWS IoT Core and SageMaker. From an implementation point of view, we need to define custom rules (as seen in *Chapter 10*) and connect to the data source of IoT Analytics using a simple query language, such as the following:

```
  SELECT wind_speed_ms FROM devices/
  device1/sensors
```

Then, we can develop an AWS IoT Analytics notebook to connect the SageMaker endpoint and IoT data.

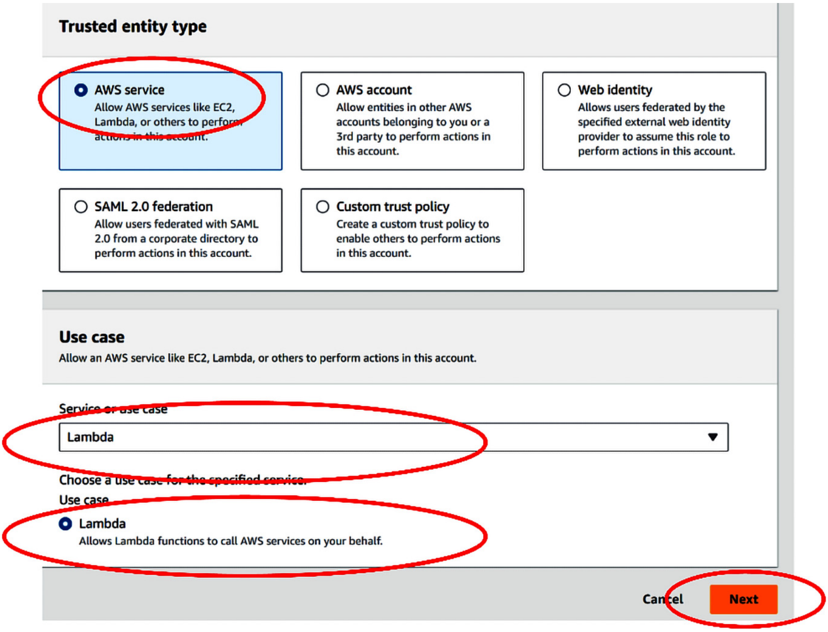
We can avoid instantiating AWS IoT Analytics; for instance, if we do not want this service and we want to reuse the model not only with AWS IoT Core, we can easily connect a straightforward AWS Lambda function to our analytical service by simply providing the URL of the final endpoint and configuring authentication policy. This may result in a more streamlined architecture that is potentially more scalable and cost-effective.

Next, we'll be connecting IoT Core to SageMaker through Lambda. Lambda is a serverless service provided by AWS to integrate services. A Lambda is a custom function developed in NodeJS, C#, PowerShell, Bash, Linux, Python, or Java. To connect IoT Hub to SageMaker through the Lambda service, we need the following three steps:

1. Define a role to connect Lambda to SageMaker
2. Create a Lambda function and test it
3. Connect IoT Core to Lambda

Creating a role

Now, access the Identity and Access Management (IAM) service (we have already studied IAM in *Chapter 10*), and from the left menu, create a new role, as shown in the following figure:



Trusted entity type

● **AWS service**
Allow AWS services like EC2, Lambda, or others to perform actions in this account.

○ **AWS account**
Allow entities in other AWS accounts belonging to you or a 3rd party to perform actions in this account.

○ **Web identity**
Allows users federated by the specified external web identity provider to assume this role to perform actions in this account.

○ **SAML 2.0 federation**
Allow users federated with SAML 2.0 from a corporate directory to perform actions in this account.

○ **Custom trust policy**
Create a custom trust policy to enable others to perform actions in this account.

Use case
Allow an AWS service like EC2, Lambda, or others to perform actions in this account.

Service or use case: Lambda

Choose a use case for the specified service.
Use case
● Lambda
Allows Lambda functions to call AWS services on your behalf.

Figure 15.38 – Creating a new role

Then, add the following permissions on the next screen:

- **`AmazonS3FullAccess`**
- **`AmazonSageMakerFullAccess`**
- **`AWSLambdaBasicExecutionRole`**
- **`IAMFullAccess`**

On the final screen, provide the role name (for instance, **`lambda_sagemaker_fullrole`**), review the policies you have added to the role, and create the role.

Creating a Lambda function

First, we need to access the Lambda service. From the search bar, locate the Lambda service, as shown here:

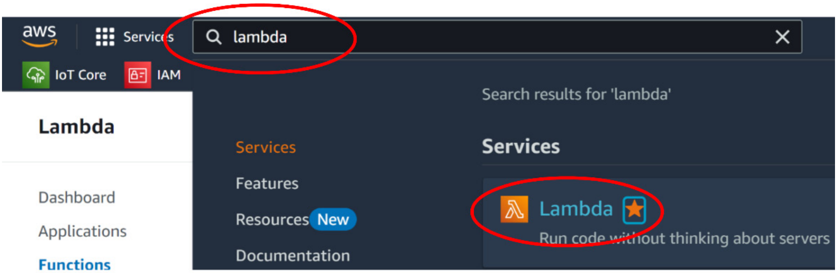
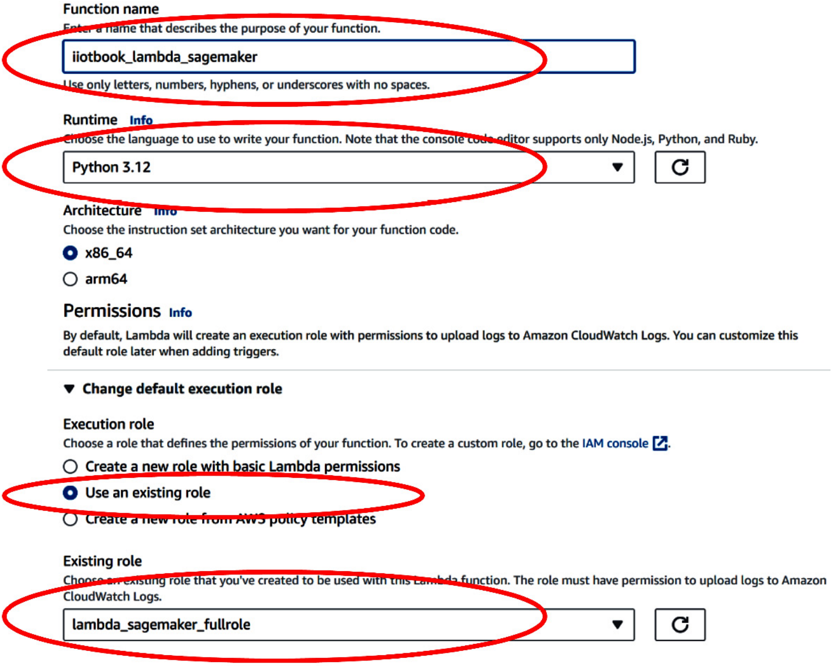

aws
Services
lambda
IoT Core
IAM
Lambda
Dashboard
Applications
Functions
Search results for 'lambda'
Services
Features
Resources New
Documentation
Services
Lambda
Run code without thinking about servers

Figure 15.39 – Accessing the Lambda service

From the left menu, go to Functions and create a function using Create Function . Under the Basic information section, provide a name for your function, select Python in the Runtime field, and set the default execution role to the role created before, as shown here:



Function name

Enter a name that describes the purpose of your function.

iiotbook_lambda_sagemaker

Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime Info

Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.

Python 3.12

Architecture Info

Choose the instruction set architecture you want for your function code.

● x86_64

○ arm64

Permissions Info

By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

▼ Change default execution role

Execution role

Choose a role that defines the permissions of your function. To create a custom role, go to the IAM console.

○ Create a new role with basic Lambda permissions

● Use an existing role

○ Create a new role from AWS policy templates

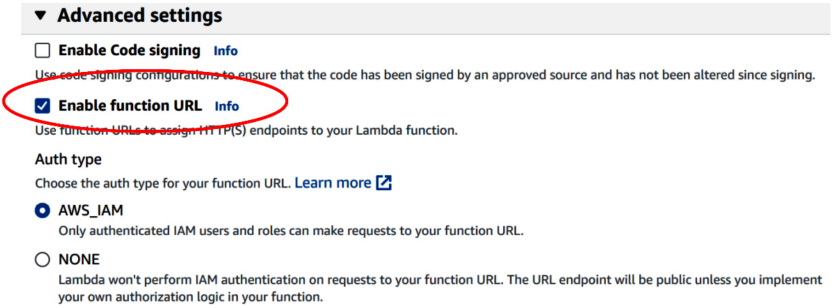
Existing role

Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.

lambda_sagemaker_fullrole

Figure 15.40 – Creating a Lambda function

On Advanced settings , click on Enable function URL :



▼ Advanced settings

☐ **Enable Code signing** Info

Use code signing configurations to ensure that the code has been signed by an approved source and has not been altered since signing.

☑ **Enable function URL** Info

Use function URLs to assign HTTP(S) endpoints to your Lambda function.

Auth type

Choose the auth type for your function URL. Learn more

● AWS_IAM

Only authenticated IAM users and roles can make requests to your function URL.

○ NONE

Lambda won't perform IAM authentication on requests to your function URL. The URL endpoint will be public unless you implement your own authorization logic in your function.

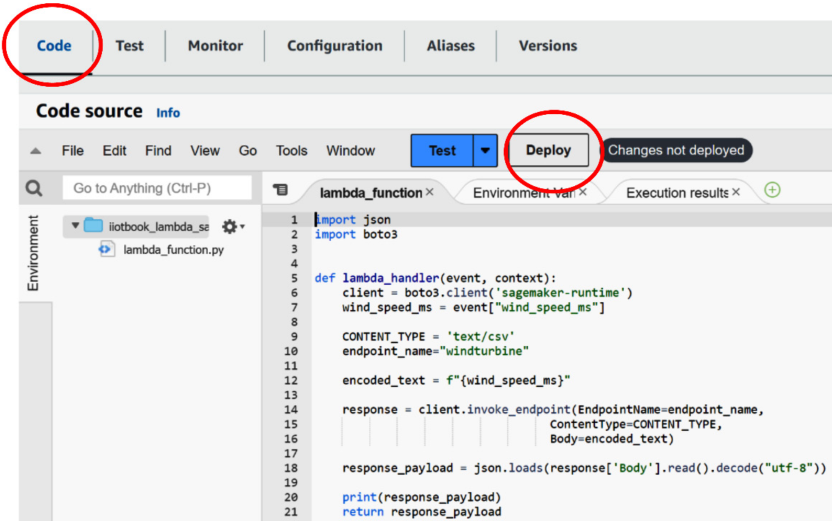
Figure 15.41 – Enabling the function URL

Click on the Create Function button and then proceed with the remaining steps and required configurations to complete the creation of your function.

On the code screen, we need to call our URL with the following code. Replace **`endpoint_name`** with the name of your model endpoint and copy and paste the following code in the source code section of the Lambda function, as shown in *Figure 15 .42* :

```
import json
import boto3
def lambda_handler(event, context):
    client = boto3.client('sagemaker-
runtime')
    wind_speed_ms =
event["wind_speed_ms"]
    CONTENT_TYPE = 'text/csv'
    endpoint_name="windturbine"
    encoded_text = f"{wind_speed_ms}"
    response =
client.invoke_endpoint(EndpointName=endpoin
ContentType=CONTENT_TYPE,
Body=encoded_text)
    response_payload =
json.loads(response['Body'].read().decode("
    return response_payload
```

Then, click on Deploy as shown in the following figure:


Code
Test
Monitor
Configuration
Aliases
Versions
Code source
Info
File
Edit
Find
View
Go
Tools
Window
Test
Deploy
Changes not deployed
Go to Anything (Ctrl-P)
Environment
iiotbook_lambda_se
lambda_function.py
lambda_function
Environment Var
Execution results

```
import json
import boto3


def lambda_handler(event, context):
    client = boto3.client('sagemaker-runtime')
    wind_speed_ms = event["wind_speed_ms"]

    CONTENT_TYPE = 'text/csv'
    endpoint_name="windturbine"

    encoded_text = f"{wind_speed_ms}"

    response = client.invoke_endpoint(EndpointName=endpoint_name,
                                      ContentType=CONTENT_TYPE,
                                      Body=encoded_text)

    response_payload = json.loads(response['Body'].read().decode("utf-8"))

    print(response_payload)
    return response_payload
```

Figure 15.42 – Deploying the Lambda function

Now, we can test our function. As shown in the previous figure, click on Test and use the following sample data to test our Lambda function:

```
{
  "wind_speed_ms": 9.37
}
```

The expected result should look like this:

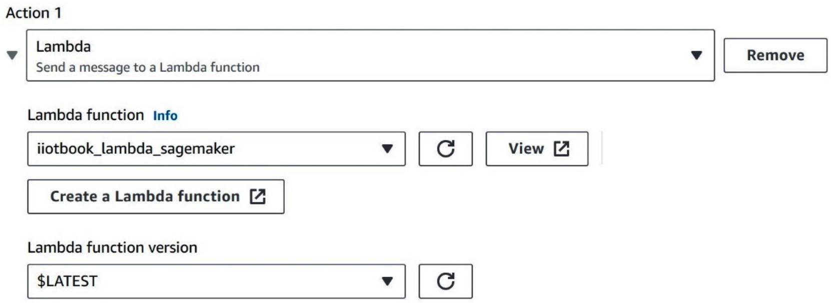
```
Response
[
    120.53847305680634
 ]
```

So, our Lambda function is up and running. Let us do the final step: connecting IoT Core and the Lambda function just created.

Connecting IOT Core with Lambda

To connect IOT Core to the just-created Lambda function, we can leverage message routing rules. We have already studied the message routing mechanism in *Chapter 10* . From the left menu of IOT Core, go to Message routing | Rules | Create Rule .

In the Action section, add the Lambda function just created, as shown in the following figure:



Action 1

Lambda
Send a message to a Lambda function

Lambda function: iiotbook_lambda_sagemaker

Lambda function version: $LATEST

Figure 15.43 – Adding an action to a rule

As source SQL, you can use the following code:

```
SELECT * FROM devices/device1/sensors
```

If everything is configured correctly, you will see the Lambda function connected to IoT Core as shown in the following figure:

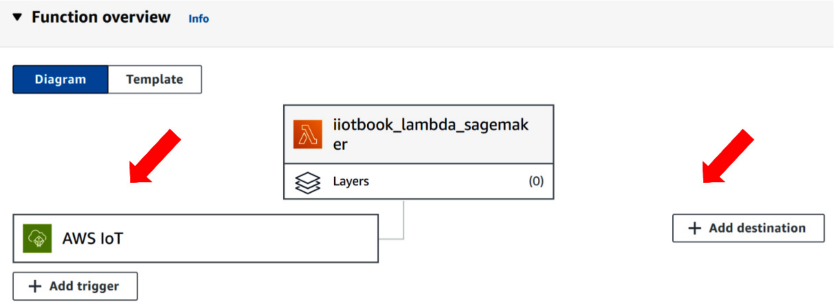

Function overview
Info
Diagram
Template
iiotbook_lambda_sagemaker
Layers
(0)
AWS IoT
+ Add trigger
+ Add destination

Figure 15.44 – AWS IoT Core connected to Lambda

At this point, you just need to connect the storage, which could be an S3 bucket or any other DB of your choice.

Cleaning resources

> *Do not forget to clean resources after experiments. From the SageMaker left menu, go to* ***Running Instances*** *and stop the instances. Moreover, do not forget to delete endpoints.*

Understanding the advanced features of SageMaker

In our minimal example, we discovered how to deploy our algorithm on SageMaker, but SageMaker supports many more functionalities other than this.

MLflow

SageMaker supports MLflow and auto-log functions. The only action to do is to start the MLflow service and activate the MLflow tracking server on the `Sklearn` processor. To start the MLflow server, create an instance as shown in the following figure and copy the Amazon Resource Name (ARN) address for future usage:

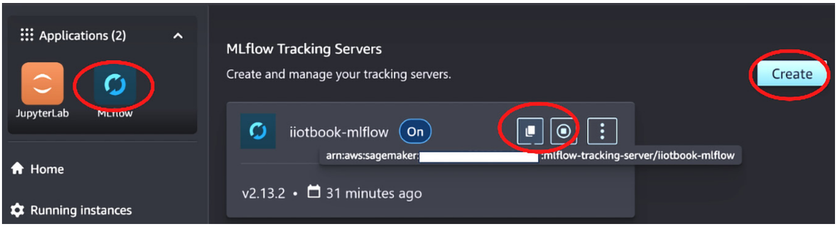
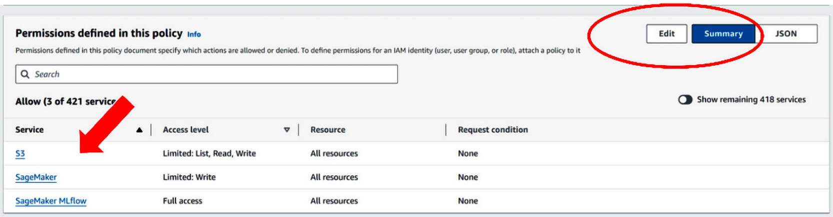

Applications (2)
JupyterLab
MLflow
Home
Running instances
MLflow Tracking Servers
Create and manage your tracking servers.
Create
iiotbook-mlflow
On
arn:aws:sagemaker:[redacted]:mlflow-tracking-server/iiotbook-mlflow
v2.13.2 •
31 minutes ago

Figure 15.45 – Starting MLflow server

We have already used AWS IAM, so from IAM, go to the left menu and then to Policies . Find your SageMaker domain's execution policy (usually called `AmazonSageMaker-ExecutionPolicy-<your id>`), click Edit , and add the following permissions, as shown in the next figure:

- `sagemaker-mlflow:*`
- `sagemaker:CreateMlflowTrackingServer`
- `sagemaker:UpdateMlflowTrackingServer`
- `sagemaker:DeleteMlflowTrackingServer`
- `sagemaker:StartMlflowTrackingServer`
- `sagemaker:StopMlflowTrackingServer`
- `sagemaker:CreatePresignedMlflowTrackingSer`



Permissions defined in this policy

Permissions defined in this policy document specify which actions are allowed or denied. To define permissions for an IAM identity (user, user group, or role), attach a policy to it

Allow (3 of 421 service

Service	Access level	Resource	Request condition
S3	Limited: List, Read, Write	All resources	None
SageMaker	Limited: Write	All resources	None
SageMaker MLflow	Full access	All resources	None

Figure 15.46 – Adding permission to publish experiments to MLflow

Then, update the policy, Now, we need to add the MLflow packages on the job and deployment instances. On a new cell, add the following line of code:

```
%%writefile src/requirements.txt
mlflow==2.13.2
sagemaker-mlflow==0.1.0
```

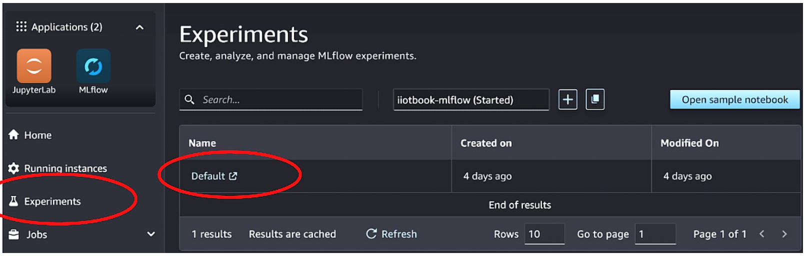
On the **`__main__`** method, add the MLflow auto log, as shown here:

```
...
    mlflow.set_tracking_uri(os.environ['MLF
    mlflow.autolog()
..
```

Finally, on the **`Sklearn`** processor, add the following environment variable:

```
...
source_dir="src",
environment={"MLFLOW_TRACKING_ARN":
the copied ARN}
...
```

The experiments will be published on the SageMaker UI, as shown in the following figure:


Applications (2)
JupyterLab
MLflow
Home
Running instances
Experiments
Jobs
Experiments
Create, analyze, and manage MLflow experiments.
Search...
iiotbook-mlflow (Started)
Open sample notebook

Name	Created on	Modified On
Default	4 days ago	4 days ago

End of results
1 results
Results are cached
Refresh
Rows 10
Go to page 1
Page 1 of 1

Figure 15.47 – Experiments on the SageMaker UI

See also the SageMaker SDK official documentation for further details: https://sagemaker-examples.readthedocs.io/ .

Auto tuning

An important feature is the ability to auto-tune hyperparameters. In our exercise, we decided to use either a single layer or the default input/output channel for the data. We can also configure different arguments and ask SageMaker to optimize our parameters. We can access these functionalities from the left menu of SageMaker.

Foundation model

Amazon SageMaker enables users to reuse a model that is already trained. This functionality can be accessed via the JumpStart option located in the left menu.

Pipelines

As with Azure ML, SageMaker lets us build data pipelines using a visual development environment. You can access it from the left menu by clicking on Pipelines .

We created and evaluated our model using SageMaker, integrating it into the IIoT workflow. Additionally, we recognized that both AWS and Azure provide highly capable and comparable solutions. This leads us to consider whether we can utilize cloud services and connect them with other platforms such as GCP or an on-premises setup. What are the benefits and drawbacks of such integrations? In the following section, we will explore these inquiries.

Understanding GCP and multi-cloud solutions

GCP is extremely focused on ML algorithms and provides a wide variety of technologies to build an analytical model. GCP has also developed its own Tensor Processing Unit (TPU) processor to speed up the adoption of ML. GCP supports ML through the Google Cloud Vertex AI and AutoML services (https://cloud.google.com/products/ai). GCP also supports several models of generative AI (GenAI), such as Gemini.

Google also offers a free development environment for ML called Colab (https://colab.research.google.com/).

In this book, we have often highlighted the differences and similarities between platforms to allow you to choose the right platform. Choosing the right cloud for our platform is a complex and risky job that is likely to have a significant impact on our business. Potential risks are represented by the learning curve, changes in the business, and costs. The following table shows common risks you can face in your organization and related countermeasures you can adopt to mitigate the risks:

Topic	Example	Countermeasure
IP protection	[illegible]	Deploy [illegible] application to protect some functionalities
[illegible]	I want to protect the IP of my analytics	Analytics developed on a Docker-protected environment
Data	I cannot export data outside my premises	Use your local cloud region [illegible] architecture

Functionalities are not clear at all	Proof of concept (POC) on an open source ready to deploy on the cloud
My business is not extensible to different assets	Microservices architecture and inter-cloud communication
My architecture is not scalable; I need to process large amounts of data	Leverage a standard cloud platform
My team is inexperienced; I need to accelerate the learning process	Leverage well-documented platforms such as open source platforms

Table 15.1 – Common risks and countermeasures when adopting an IIoT architecture

One option is to use inter-cloud communication, or Docker, Jupyter, MLflow, and Python, to be as agnostic as possible with regard to the cloud. Indeed, staying open to common standards and platforms is the most advantageous approach. In this regard, throughout our journey, AWS and Azure have demonstrated their ability to provide excellent scalability, openness, and integration features.

Summary

In this chapter, we discovered how to deploy analytics on Azure ML and Amazon SageMaker, using the algorithms developed in *Chapters 13* and *14* . We looked at the differences between the deployment methodologies of Azure and AWS, and we also explored the new trend of IoT analytics.

This chapter was the final chapter of our journey into the IIoT. We started in *Chapter 1* by looking at the differences between the IoT and the IIoT. The first four chapters talked about the most important data sources in the industrial sector and the differences between them. We looked at the Open Platform Communications Unified Architecture (OPC UA) standard and learned how it is the de facto standard in the industrial sector. We also understood the differences between data polling and data subscription. In *Chapter 7* , we explored a general-purpose architecture of IIoT platforms and implemented three different prototypes in the chapters that followed: open source, Azure, and AWS platforms. We learned that the IIoT is predominantly made up of time series data, but also logs and static data such as asset information. In *Chapter 13* , we discovered the most important classes of analytics: diagnostic, predictive, prescriptive, and prognostic. Finally, in *Chapters 14* and *15* , we introduced AI, and we explored the concept of a digital twin and the process of its development. We also traced the most important steps to build analytics and the three ways to deploy analytics: using the hot path, the cold path, and on the edge or the cloud.

Before concluding our journey into the world of the IIoT, we would like to recall some key points. The IIoT is primarily a collection of technologies, therefore we strongly recommend maintaining a modular and multi-modal architecture. In fact, a new player will appear in the IIoT field: AI will be increasingly predominant and a decisive ally of the IIoT in processing data, and the business will ask you to introduce it at every level: in data cleaning, in understanding and monitoring assets, and in knowledge management.

Therefore, from an architectural point of view, it is essential to bear in mind that the basic idea of microservice architecture and the SOLID principles remain valid. The IIoT platform is not a monolithic application but, instead, an ecosystem of different bricks. You should design every brick as a single autonomous component with standard input and output. If you do this, you should not experience insurmountable issues.

Index

As this ebook edition doesn't have fixed pagination, the page numbers below are hyperlinked for reference only, based on the printed edition of this book.

A

Access Control List (ACL) 125 , 128

accurate ML model, with Azure notebook 455

  building, on Azure ML notebook 458

  EDA 457

  packaging and deploying 459 , 460

  problem statement and dataset 455 , 457

actuators 45 , 53 , 54

Adam 419

address space 90

advanced analytics 177

advanced controllers 60

Advanced Encryption Standard (AES) 92

Advanced Message Queuing Protocol (AMQP) 39 , 102 , 105 , 225

Advanced Process Control (APC) 40

Airflow 225

- analytics 205 - 207
- batch analytics, developing 201
- connecting, to InfluxDB 207 - 211
- installing, for production 201 , 202
- installing, on base host 203
- online analytics, developing 211 - 216

Airflow architecture

- meta-database 204
- scheduler 204

  - webserver 204
  - worker 204
- Airflow UI
  - connecting to 203
- Amazon Resource Name (ARN) 482
- Amazon SageMaker
  - accessing 464 - 466
  - analytics, implementing on 464
  - auto tuning 484
  - features 482
  - foundation model 484
  - MLflow 482 - 484
  - model, consuming from AWS IoT Core 476 , 477
  - pipelines 485
  - S3 bucket, preparing for data storage 466 - 468
  - used, for developing digital twin 469

Amazon Web Services (AWS) 227 , 438

analogic signals 53

analog sensors 46

Analog to Digital Converter (ADC) 51 , 52

  signals 53

analytics, IIoT 175

  descriptive analytics 175

  diagnostic analytics 175

  predictive analytics 175

  prescriptive analytics 175

anomaly detection 376

  collective anomalies 377

  contextual anomalies 377

  in practice 391 - 398

  point anomaly 377

  with supervised ML 402 - 406

- with unsupervised ML 398 - 402

Ansys 436

Apache ActiveMQ 225

Apache Airflow 201

Apache Flink 225

Apache NiFi 225

Apache Spark 225

application programming interface (API) 83

application proxy firewall 129

artificial intelligence (AI) 4 , 381 , 414

artificial intelligence of things (AIoT) 15 , 381 , 414

- drawbacks 15

Artificial Neural Network (NN) 382

ASQL

- features 345

assembler 55

asset classes 175

asset instance 175 , 217

asset model 76 , 217 , 298

  building, with Neo4j 218 - 220

  characteristics 77

asset registry 174 , 175 , 222 , 273

  building, to store asset information 217

  integrating, into data flow 222 , 223

attributes 90

automation networks 65

Autonomous Guided Vehicles (AGVs) 25

Autoregressive Integrated Moving Average (ARIMA) model 399

availability 224

AWS architecture 229

AWS Athena 279

AWS command-line interface client
    installing 238 , 239
AWS data flow architecture 278
AWS Free Tier account
    registering 231 - 238
AWS IoT 229 , 230
    components 229 , 230
AWS IoT Analytics 278 , 279 , 289
    AWS Athena 279
    AWS IoT Events 289 - 293
    AWS SageMaker 279
    used, for implementing hot data path 279 , 281 - 288
    working 281
AWS IoT Core
    connecting, with Lambda function 481 , 482

  digital twin model, consuming 476
AWS IOT Edge 105
AWS IoT Events 289 - 294
AWS SageMaker 279 , 435
AWS SDK
  using 248 - 254
Azure Data Explorer 362 - 369
  reference link 369
Azure IIoT architecture 338
  data flow, setting 338
  visualizations, building 362
Azure IoT
  ecosystem 310 , 311
  reference link 312
  registering for 311 , 312
  supported methods 312

Azure IoT Edge 105 , 317 , 322 , 463

  applying, to industrial sector 326 - 328

  custom module, building 328 , 329

  features 323

  starting 324 , 325

Azure IoT Hub 312 - 315

  Azure IoT Edge 322 - 324

  custom edge, building 317

  messages, monitoring through IoT Hub metrics 330

  new device, registering 315 - 317

  query language, of device twins 329

Azure ML service

  analytical model, developing 446

  Azure IoT Edge 463

  IoT Hub integration 462 , 463

- resources, cleaning 461 , 462
- starting with 448
- wind turbine digital twins, developing with Azure ML 449
- working with 446 , 447
- workspace 449

Azure Stream Analytics (ASA) 329 , 345 , 355 , 356

- user-defined functions 352 - 355
- working with 346 - 352

Azure Synapse 357

- and Cosmos DB, working with 358 - 361
- prerequisites 357
- workspace, creating 357 , 358

B

base host

- Airflow, installing on 203

batch analytics 388

  developing, with Airflow 201

batch processes 24 , 25

big data

  versus IoT analytics 225

big data analytics 177

  reference link 177

binary sensors 46

Business Intelligence (BI) 375

bus loop 64

C

Cambridge Analytica scandal 123

Capital Expenditure (CAPEX) 7

Carrier Sense Multiple Access/Collision Detection (CSMA/CD) 65

centralized access/polling method 65

Central Processing Unit (CPU) 54

CIM pyramid 25 , 26

  levels 28 - 30

  logical schema 27

  three-level structure 30

CIM pyramid architecture

  devices and networks 31 - 34

  networks 34 , 35

closed loop 64

cloud

  disadvantages 101

  implementing, barriers in industrial plants 106

cloud analytics 389

Cloud-based TSDBs 173

CoAP 105

  URL 105

Colab

  reference link 485

cold path 171 , 189

command-line interface (CLI) 238

commercial IIoT platforms

  AWS architecture 229

  AWS command-line interface client, installing 238 , 239

  AWS Free Tier account, registering 231 - 238

  AWS IoT 229 , 230

  need for 228 , 229

Commercial off-the-Shelf (COTS) 6 , 61

Complex Instruction Set Computing (CISC) 55

Component Object Model (COM) 81

Computer-Aided Design (CAD) 26

Computer-Aided Engineering (CAE) 26

Computer-Aided Process Planning (CAPP) 26

computer-integrated manufacturing (CIM) 17 , 170

computerized maintenance management system (CMMS) 174

Computer Numerical Controls (CNCs) 29 , 32

condition-based analytics 388

Condition-Based Maintenance (CBM) 380

Constrained Application Protocol (CoAP) 102 , 190

containers 166

Containers as a Service (CaaS) 169

continuous integration and continuous delivery (CI/CD) 323 , 449

continuous processes 24

control and measurement system model 21 - 23

control bus 69

controlled value (CV) 45

controllers 54

control module 78

control network (CN) 66 , 127

control-network-segregation architectures 130

- firewall, with DMZ 132
- firewall, with DMZ and VLAN 134 , 135
- network separation, with a single firewall 130 , 131
- paired firewall, with DMZ 133

converters 48

- Analog to Digital Converter (ADC) 51 , 52
- Digital to Analogical Converter (DAC) 49 , 50

Cosmos DB 311 , 338 , 341 , 342

create, read, update, and delete (CRUD) 219

custom edge, Azure IoT Hub

- building 317

- building, with Node-RED 317 - 320
- building, with Python 320
- starting, from local PC 322

custom function, with Lambda

- reference link 286

custom signal

- adding 148 , 149

Cyber Physical Systems (CPSs) 12

cycle 204

Cypher 219

D

Dagster 225

data acquisition interval

- factors 70

data collection

  - polling 69
  - unsolicited 69
- data-driven surrogate model 60
- data flow
  - setting 338
- data flow, from MQTT to InfluxDB
  - starting 195 , 196
  - testing 197
- data-gathering techniques 107
- data Historian 71
- data management use cases
  - Azure Synapse 357
  - data processing 339 , 345
  - data storage 338 , 339
  - data transformation 339
  - Machine Learning analytics 361

datapoints 71

data processing

  Advanced Stream Analytics 355 , 356

  Azure Stream Analytics, using 345

  Azure Stream Analytics, working with 346 - 352

  user-defined function, on ASA 352 - 355

data-processing analytics

  advanced analytics 177

  big data analytics 177

  excursion monitoring analytics 176

data storage 294 , 296 - 339

  Blob Storage 339 , 341

  Cosmos DB 341 - 345

  storage browser 339 , 341

  Time Series Insights (TSI) 345

deep learning (DL) 411 , 417 - 419

  - PyTorch 420
  - TensorFlow 420
- Deep Packet Inspection (DPI) firewalls 129
- Defense in Depth (DiD) approach 123
- demilitarized zone (DMZ) 114 , 127
- denial of service (DoS) 92
- descriptive analytics 376
  - condition monitoring 376
  - health monitoring 376
  - KPI monitoring 376
- detector model 293
- development environment
  - preparing 421
- device bus 69
- device provisioning service (DPS) 312
- diagnostic analytics 376

  anomalies, discovering 378

  anomalies, root cause 378

  anomaly detection 377 , 378

DiD strategy 124

  awareness, creating among people 124

  firewalls 127 , 128

  operating modes and procedures 125

  technology 125

  using, in industrial control system environment 126 , 127

digital sensors 46

Digital Signal Processing (DSP) 57

digital signals 53

Digital to Analogical Converter (DAC) 49

  errors 50

  properties 50

digital twin 177

digital twin, developing with SageMaker 469

  dataset 470

  EDA 471

  environment 470

  model, building 471 - 475

  packaging and deploying 475

  problem statement 469

digital twin model 379

digital twin model, consuming from AWS IoT Core 476

  Lambda function, creating 478 - 481

  role, creating 477

digital twins 411 - 413

  advanced modeling 436

  best practices 413

    developing 421
    development environment, preparing 421
    implementing, in IIoT 413 , 414
    platforms 434 - 436
    RUL of 100 engines, evaluating 422
    RUL of 100 engines, evaluating steps 422 - 431
    used, in following sectors 412 , 413
    wind turbine, monitoring 431 - 434
directed acyclic graph (DAG) 201 , 223
directed graph 204
discrete processes 25
Distributed Component Object Model (DCOM) 81
distributed control systems (DCSs) 17 , 43 , 61 , 82 , 109 , 173
    using, conditions 61
Docker 167

- commands 168
- microservice, building with 167

Docker Build

- using 182 - 185

Docker Compose 168

- URL 168
- using 182 - 185 , 191

Docker Desktop Edition

- URL 167

dual-homing 130

E

edge 100 , 151

- connecting, options to industrial data sources 112 , 113

edge architecture 102

edge client

- installing 151 - 153

edge computing 103

- features 104

edge device 39 , 40

edge gateway 100 , 102

- features 103

edge internet protocols 105

edge, on fieldbus setup 113

- advantages 114
- disadvantages 114

edge, on OPC DCOM 115

- advantages 116
- disadvantages 116

edge, on OPC-Proxy 116 , 117

- advantages 118
- disadvantages 118

edge, on OPC UA
  advantages 119 , 120
  disadvantages 120
edge tools 104
  features 104
Elasticsearch 173 , 226
embedded microcontrollers 56
encapsulation technique 64
encoder modules 59
end-to-end IoT flow
  implementation with InfluxDB, Node-RED, and OPC UA 178 - 182
Enhanced Performance Architecture (EPA) 67
enterprise resource planning (ERP) 13 , 17 , 26 , 74 , 174 , 375
equipment module 78

Event Hubs

    reference link 315

excursion management system (EMS) 293

excursion monitoring analytic (EMA) 176 , 293 , 345

Explainable AI (XAI) 383

Exploratory Data Analysis (EDA) 383

Extract, Transform, Load (ETL) orchestrator system 225

F

fieldbus

    edge, securing on 136 - 138

field/fieldbus networks 66 , 68

    advantages 67 , 68

field network 66

File Transfer Protocol (FTP) 129

final control component 54

firewalls

  application proxy firewall 129

  Deep Packet Inspection (DPI) firewalls 129

  goals 129

  packet filtering firewall 128

  stateful inspection firewall 129

firewall, with DMZ 132

firewall, with DMZ and VLAN 134 , 135

fog computing

  versus IIoT edge 102

frames 63

Functions as a Service (FaaS) 170

G

Gaussian process regression (GPR) 430

general AI (AGI) 414

generative adversarial networks (GANs) 415

generative AI (GenAI) 408 , 420 , 485

Google Cloud Platform (GCP) 435 , 485

gradient descent (GD) 418

Grafana 197 , 300 - 307 , 365 - 368

- data, visualizing 197 , 198 , 200

graph database 218

Graphic Process Unit (GPU) 390

Graphite 192

- reference link 192

Greengrass 105 , 268

Greengrass V2 254 - 268

H

Heat Dissipation Failure (HDF) 403 , 409

Historians 17 , 45 , 71 - 74 , 111

  - advantages 111
  - categories 74
  - characteristics 73
  - disadvantages 112

Historian-to-Historian 73

Historical Data Access (HDA) 112

hot data path

  - implementing, with AWS IoT Analytics 279 - 288

Hot path 171 , 189 , 345

HTTP 105

HTTPS 105

Human Machine Interface (HMI) 45 , 70

Hypertext Transfer Protocol (HTTP) 129

Hypertext Transfer Protocol Secure (HTTPS) 39

I

IBM Cloud 435

Identity and Access Management (IAM) 477

IIoT analytics 374

- descriptive analytics 376
- diagnostic analytics 376
- predictive analytics 378
- prescriptive analytics 379
- role of infrastructure 386
- use cases 375

IIoT analytics, building 383

- dataset acquisition 384
- deploying 385
- development workflow 384
- EDA 385
- model, building 385

  monitoring 385
  packaging 385
  problem statement 384
IIoT analytics, deploying 386 , 387
  advanced analytics 390
  cloud analytics 389
  condition-based analytics 388
  interactive analytics 389
  on-controller analytics 389
  on-edge analytics 389
  streaming analytics, versus batch analytics 387 , 388
IIoT analytics, in practice 391
  anomaly detection, in practice 391
  anomaly detection, with supervised ML 402
  anomaly detection, with unsupervised ML 398

  - predictive production analytics, in practice 406
  - prescription, with language models 408 , 409
- IIoT analytics technologies 380
  - model-based analytics 381
  - rule-based analytics 381
- iiot-book-2-net network 219
- IIoT data 226 , 436
- IIoT data flow 35
  - edge device 39 , 40
  - edge, securing on fieldbus 136 , 137
  - edge, securing on OPC DCOM 138 - 140
  - edge, securing on OPC Proxy 140 - 142
  - edge, securing on OPC UA 142 , 143
  - implementing 106
  - in cloud 40
  - in factory 36 - 38

    - OPC UA, securing on controller 144 , 145
    - securing 135
- IIoT data flow in factory 44 , 45
    - examples 45
- IIoT data sources 107
- IIoT edge 100
    - versus fog computing 102
- IIoT features
    - aws.greengrass.DiskSpooler 268
    - device shadows 268
    - Lambda runtimes 268
- IIoT platform 164
    - analytics 175
    - architectures 164
    - end-to-end data flow 171
    - standard IIoT flow 170 - 172

industrial adapter 103

industrial control system environment

  DiD, using 126

Industrial Control System (ICS) 35 , 126

industrial IoT (IIoT) 11 , 227 , 277

  digital twins, implementing 413

  features 13

  industry environments and scenarios covered 15

  use cases 12

  versus IoT 13 , 14

industrial process 18

  automation 19 - 21

  control and measurement systems 21 - 23

  production process 19

  transformation, from raw materials to products 18 industrial processes, types 23

  batch processes 24

  continuous processes 24

  discrete processes 25

  semi-continuous processes 25

industrial protocols 62

  automation networks 65 , 66

  field/fieldbus networks 66 - 69

  OSI/ISO model 62 - 65

Industry 4.0 11

Industry 5.0 12

InfluxDB 173

  Airflow, connecting to 207 - 211

  time-series data, storing 191 , 192

InfluxDb, and MQTT Data Broker

  connecting 193 , 194

information network 66

Information Technology (IT) 126

Input and Output (I/O) 55

integrated control system (ICS) 109

Intellectual Property (IP) 14 , 35

interactive analytics 389

International Electrotechnical Commission (IEC) 21

Internet of Content 4

Internet of Services 4

Internet of Things (IoT) 4 , 414

- definition 5
- enabling factors 6 - 8
- history 4 - 6
- key technologies 10
- use cases 9 , 10

  versus IIoT 13 , 14

Intrusion Detection System (IDS) 125

Intrusion Prevention System (IPS) 125

I/O modules 59

IoT adapter 103

IoT analytics

  applications 14

  data-driven (AI and machine learning 15

  hybrid model 15

  physics-based category 15

  versus big data 225

IoT Analytics framework 288

IoT Central 369

IoT Core 239 , 240

  AWS MQTT endpoint, identifying 240

  AWS SDK, using 248 - 254

  - Greengrass V2 254 - 268
  - Node-RED, connecting 241
  - SiteWise 269 - 275
- IoT edge
  - versus IIoT edge 101
- IoT hub 40
- IoT Hub metrics
  - sent messages, monitoring through 330
- ISA-95 equipment entities 77 , 78
- items 61

J

- Joblib
  - reference link 463
- Jupyter 374
  - URL 389

K

Kedro 225

Key Performance Indicators (KPIs) 40 , 375

Kibana 226

Kubeflow 225

  URL 446

Kusto Query Language (KQL) 362

L

Laboratory Information Management Systems (LIMSs) 75

large language model (LLM) 408 , 420

learning rate (LR) 419

Least Significant Bit (LSB) 49

local area network (LAN) 103

Low-Power Networks (LPNs) 7

Low-Power Wide-Area Networks (LPWANs) 7

Luigi 225

M

MaaS

  Airflow integration 445 , 446

  developing, with MLflow 439

machine language 55

machine learning (ML) 411

Machine Learning (ML)

  analytics 40 , 310

Machine Learning Operability (MLOps) 383

manufacturing execution system (MES) 13 , 17 , 26 , 75 , 223

marshaling 84

MATLAB

  reference link 436

measurement and actuator chain 45

Message Queuing Telemetry Transport (MQTT) 39 , 102 , 105 , 189

MES system

- advantages 75
- features 75

micro-batch analytics 388

microcontroller 54

- DSPs 57
- embedded microcontrollers 56
- features 56
- groups 56
- used, in systems 55
- versus microprocessors 55
- with external memory 57

microservice-oriented technologies 169

microservices 165 , 166

  - building, with Docker 167
- Microsoft Azure 435
- mini-batch GD 418
- ML algorithms 414
  - development 415 - 417
  - reinforcement learning (RL) 415
  - scikit-learn 417
  - supervised learning 414
  - unsupervised learning 415
- MLflow
  - MaaS, developing with 439
  - starting 439 , 440
  - URL 439
- ML model
  - building 441 - 443
- ML model development 441

  - exploratory data analysis (EDA) 441
  - packaging and deploying 443
  - problem statement and dataset 441
  - serving, with MLflow API and FastAPI 443 , 444
  - serving, with MLflow server 444 , 445
- ML operations (MLOps) 439
- model-as-a-service (MaaS) 438
- model-based analytics 381
  - data-driven models 381 , 382
  - data-driven versus physics-based models 382
  - hybrid model 382
  - physics-based models 381
  - reinforcement learning 382
  - semi-supervised learning 382
  - supervised learning 382
  - unsupervised learning 382

Model Predictive Control (MPC) 60

MongoDB 175

Mosquitto 189

- as data gateway 190 , 191
- URL 190

Most Significant Bit (MSB) 49

multifactor authentication (MFA) 235

N

natural language processing (NLP) 414

Neo4j 175

- asset model, building 218 - 220
- URL 218
- working with 221 , 222

network firewalls 128

network module 59

neural networks (NNs) 417 , 418

Neural Process Unit (NPU) 390

Node-OPCUA

- reference link 148

Node-RED 105 , 153 , 195

- connecting 241 , 245 - 247
- custom edge, building with 317 - 320
- flow, building 155 - 159
- implementation 240
- installing 153
- running, on Docker 154
- security policy, setting 241 - 245

Node-RED edge 153

novelty detection 402

NVIDIA Omniverse 435

O

Object Linking and Embedding (OLE) for Process Control (OPC) 81

Omni digital twin model 413

on-controller analytics 389

One-Class Support Vector Machine (OCSVM) 399

on-edge analytics 389

Online Analytical Processing (OLAP) 75

online analytics

  developing, with Airflow 211 - 216

OPC Classic 83 - 86

  data flow, implementing consideration 87 , 88

  goal 86

  limitations 85

  reference link 83

  server groups 87

  - working 86

- OPC Data Access (DA) 87
- OPC DCOM
  - edge, securing on 138 - 140
- OPC Foundation
  - URL 148
- OPC Proxy
  - edge, securing on 140 - 142
- OPC UA 88
  - edge, securing on 142 - 144
  - goal 88
  - sessions 91
  - working 88
- OPC UA data exchange 95 , 98
  - read service parameters 96
  - subscription, managing 97

  - subscription parameters 97
  - write services parameters 96
- OPC UA information model 89
  - method node class 90
  - object node 90
  - principles 89
  - reference 90
  - variable node 90
  - view 91
- OPC UA notifications 98 , 99
- OPC UA on controller 120 , 122
- OPC UA sample server
  - starting 148
- OPC UA security model 91 - 95
  - endpoint features 92
- OPC UA Semantic 91

OPC UA Server
    securing 159
OPC UA Server Node.js 148
    custom signal, adding 148 , 149
OPC UA Simulation Server 148
OPC Unified Architecture (UA) 190
OPC XML DA 88
Open Container Initiative (OCI) 167
open loop 64
Open Neural Network eXchange (ONNX) 463
Open Platform Communications (OPC) 38 , 147 , 190
    discovering 82 , 83
open source IIoT architecture 189
open source platforms
    Apache ActiveMQ 225

  - Elasticsearch 226
  - Kibana 226
  - RabbitMQ 225
  - Redis 226
- open source solutions
  - reasons, for developing 188 , 189
- open source technologies, for analytics
  - Apache Flink 225
  - Apache NiFi 225
  - Apache Spark 225
  - Dagster 225
  - Kedro 225
  - Kubeflow 225
  - Luigi 225
- Open System Architecture (OSA) 390
  - condition monitoring 390

  - data acquisition 390
  - decision support 390
  - health assessment 390
  - prognostics 390
  - signal processing 390
- OpenTSDB 173
- OpenUSD 435
- Operational Expenditure (OPEX) 8
- operational firewalls 114
- Operational Technology (OT) 8
- optimal controllers 60
- options, for data visualization
  - Azure Data Explorer 362 , 369
  - Grafana 365
  - IoT Central 369
  - Power BI 369

OSI/ISO model 62

  features 63 , 64

  functional layers 63

OSIsoft PI 173

out-of-order data 356

Overall Equipment Effectiveness (OEE) 224

overfitting 416

Overstrain Failure (OSF) 403

P

packet filtering firewall 128

packets 128

paired firewall, with DMZ 133

partition key 352

performance 224

perimeter network (PN) 132

Personal Area Networks (PANs) 7

Physics-Encoded Neural Networks (PeNNs) 382

Physics-Guided Neural Networks (PgNNs) 382

Physics-Informed Neural Networks (Pi-NNs) 382

Pickle 463

Platform-as-a-Service (PaaS) 166

platforms

    comparing 369

Power BI 369

Power Failure (PWF) 403

predictive analytics 378

    other predictive analytics 379

    prognostic analytics 378 , 379

predictive production analytics

    in practice 406 - 408

prescriptive analytics 379

- Condition-Based Maintenance (CBM) 380
- production optimization analytics 380
- with language models 408 , 409

Probability Density Function (PDF) 378

Process Control Network (PCN) 127

process control system 75

process database 70

process value (PV) 46

producer/consumer access method 65

production
- Airflow, installing for 201 , 202

production optimization analytics 380

prognostic analytics 378

programmable logic controllers (PLCs) 17 , 43 , 45 , 57 , 82 , 108
- advantages 108

  - disadvantages 108 , 109
  - encoder modules 59
  - I/O modules 59
  - Model Predictive Control (MPC) 60
  - network module 59
  - processor module 58
  - remote I/O module 59
  - servo modules 59
- Prometheus 173
- Proof of concept (POC) 485
- Prophet 406
- Proportional-Integral-Derivative (PID) controllers 29 , 59
- proposed platform
  - pros and cons 224
- Prosys OPC server

    - installing 149

Prosys OPC UA server 149

Prosys OPC UA Simulator 149

    - measures, simulating 150 , 151
    - reference link 149

Protocol Data Unit (PDU) 64

Public Key Infrastructure (PKI) 125

Python

    - code, building 320 , 321
    - containers, building 321 , 322
    - custom edge, building with 320

PyTorch 420

Q

quantization error 52

quantized signals 53

R

RabbitMQ 225

Radio-Frequency Identification (RFID) 5

Random Failures (RNF) 403

raw data 72

Real-Time Operating System (RTOS) 7 , 390

Redis 226

Reduced Instruction Set Computing (RISC) 55

reinforcement learning (RL) 60 , 382 , 415

remaining useful life (RUL) 378 , 411

remote I/O module 59

remote procedure call (RPC) 84

Remote Terminal Unit (RTU) 32 , 43

Retrieval Augmented Generation (RAG) system 408

Riak TS (RTS) 173

ring loop 64

root mean squared error (RMSE) function 429

rule-based analytic

    pros and cons 216 , 381

RUL, of jet engine

    evaluating, steps 422 - 431

S

SA-88 extensions 78 - 80

safety instrumented systems (SISs) 82

sampled signals 53

SCADA systems 110

    pros and cons 110 , 111

scan time 58

scikit-learn 417

    cheat sheet 417

second digital platform 5

semi-continuous processes 25

semi-supervised learning 382 , 414

sensor 45 , 46

    characteristics 47 , 48

sensor bus 68

Sequences of Events (SOEs) 38

Service Bus 330

    sent message, displaying 330 - 335

service-oriented architecture (SOA) 165

servo modules 59

set point (SP) 45

signal conversion 54

single responsibility pattern 169

single source of truth 34

SiteWise 269 - 276

software-as-a-service (SaaS) 169 , 438

software development kits (SDKs) 104

Standard SQL 175

star topology 64

stateful inspection firewall 129

stochastic gradient descent (SGD) 418

store and forward functionality 72

stream analytics 388

supervised learning 382 , 414

supervisory control and data acquisition (SCADA) 17 , 43 , 45 , 69 , 71

system database 61

T

tags 61 , 71

Telegraf 193

  macro-functionalities 193

  reference link 193

TensorFlow 420

tensor processing units (TPUs) 420 , 485

time division method 65

Timescale 173

time-series data

  storing, on InfluxDB 191 , 192

time-series database (TSDB) 40 , 71 , 171

  Cloud-based TSDBs 173

  Elasticsearch 173

  InfluxDB 173

  OpenTSDB 173

  OSIsoft PI 173

  Prometheus 173

  Riak TS (RTS) 173

  Timescale 173

  Uniformance PHD 173

Time Series Insights (TSI) 345

time-series technologies 172

Timestream 294

token 65

token access method 65

token bus 65

Tool Wear Failure (TWF) 403

topology 64

  examples 64

transducer 46

transmission media

  coaxial cable 64

  optical fiber 64

  powerlines 64

  wireless networks 64

Trusted Platform Module (TPM) 40

U

UaExpert

  reference link 151

Uncertainty Quantification (UQ) 378

underfitting 416

Unified Architecture (UA) 38 , 81 , 147

  reference link 83

Uniformance Process History Database (PHD) 173

unsupervised learning 382 , 415

user-defined function (UDF) 352

user level 69

V

validation authority (VA) 94

variational autoencoder (VAE) 400 , 420

virtual machine (VM) 167

visualization 299 , 307

- building 362
- Grafana 300 - 307
- workforce users 299

Von Neumann architecture 55

W

wide area network (WAN) 103

windowing 355

Windows PowerShell 251

Windows Subsystem for Linux (WSL) 324

wind turbine

- monitoring 431 - 434

wind turbine digital twins development, with Azure ML 449

- accurate model, developing with Azure notebook, using 455

  - Azure notebook, using 453 , 454
  - building, from local Jupyter instance 460 , 461
  - Jupyter Notebook, using 451 - 453
  - model, refreshing 450
- workforce users 299

Y

- Yet Another Resource Negotiator (YARN) cluster 225

Z

- Zero-Order Hold (ZOH) 49

packtpub.com

Subscribe to our online digital library for full access to over 7,000 books and videos, as well as industry leading tools to help you plan your personal development and advance your career. For more information, please visit our website.

Why subscribe?

- Spend less time learning and more time coding with practical eBooks and Videos from over 4,000 industry professionals
- Improve your learning with Skill Plans built especially for you
- Get a free eBook or video every month
- Fully searchable for easy access to vital information
- Copy and paste, print, and bookmark content

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files

available? You can upgrade to the eBook version at packtpub.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at customercare@packtpub.com for more details.

At www.packtpub.com , you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on Packt books and eBooks.

Other Books You May Enjoy

If you enjoyed this book, you may be interested in these other books by Packt:

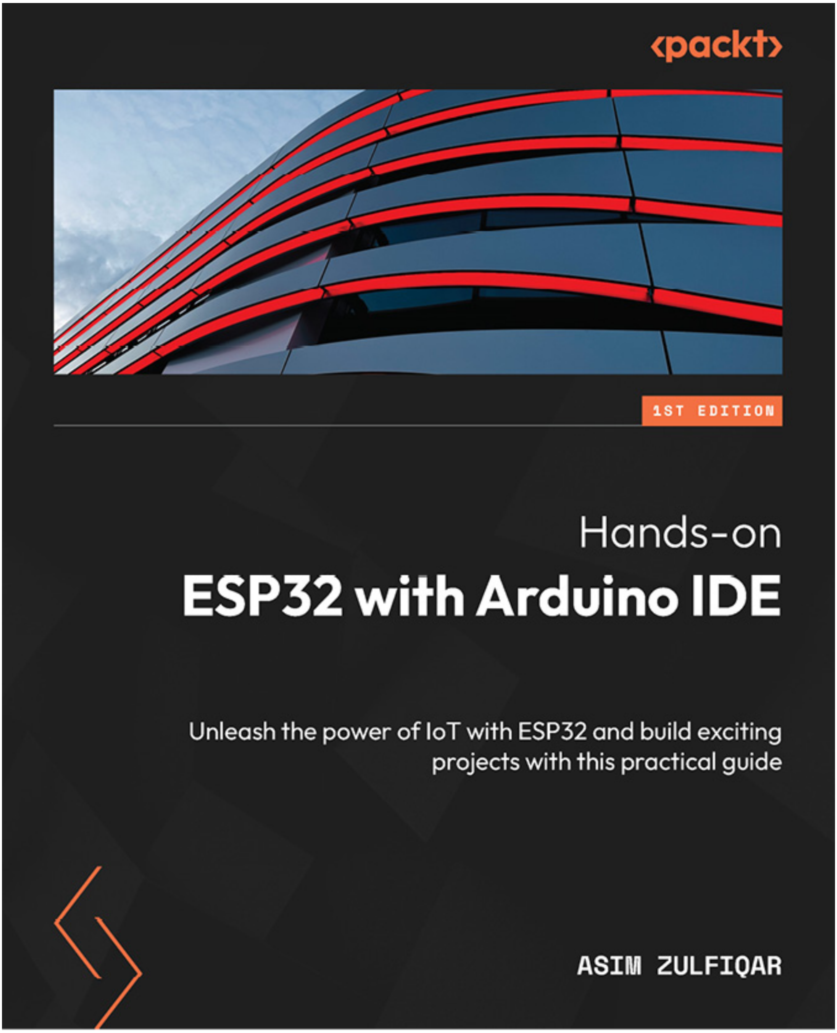
packt
1ST EDITION
Building Secure Automotive IoT Applications
Developing robust IoT solutions for next-gen automotive software
DR. DENNIS KENGO OKA | SHARANUKUMAR NADAHALLI
JEFF YOST | RAM PRASAD BOJANKI
Foreword by Dr. André Weimerskirch, VP Product Integrity

Building Secure Automotive IoT Applications

Dr. Dennis Kengo Oka, Sharanukumar Nadahalli,
Jeff Yost, Ram Prasad Bojanki

ISBN: 978-1-83546-550-9

- Explore the current automotive landscape and IoT tech trends
- Examine automotive IoT use cases such as phone-as-a-key, predictive maintenance, and V2X
- Grasp standard frameworks such as classic and adaptive AUTOSAR
- Get to grips with vehicle diagnostic protocols such as UDS, DoIP, and SOVD
- Establish a secure development process and mitigate software supply chain risks with CIAD, RASIC, and SBOM
- Leverage ASPICE and functional safety processes for industry standards compliance
- Understand how to design, develop, and deploy an automotive IoT application



Hands-on ESP32 with Arduino IDE

Asim Zulfiqar

ISBN: 978-1-83763-803-1

- Understand the architecture of ESP32 including all its ins and outs

- Get to grips with writing code for ESP32 using Arduino IDE 2.0
- Interface sensors with ESP32, focusing on the science behind it
- Familiarize yourself with the architecture of various IoT network protocols in-depth
- Gain an understanding of the network protocols involved in IoT device communication
- Evaluate and select the ideal data-based IoT protocol for your project or application
- Apply IoT principles to real-world projects using Arduino IDE 2.0

Packt is searching for authors like you

If you’re interested in becoming an author for Packt, please visit authors.packtpub.com and apply today. We have worked with thousands of developers and tech professionals, just like you, to help them share their insight with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Share Your Thoughts

Now you've finished *Hands-On Industrial Internet of Things* , we'd love to hear your thoughts! If you purchased the book from Amazon, please click here to go straight to the Amazon review page for this book and share your feedback or leave a review on the site that you purchased it from.

Your review is important to us and the tech community and will help us make sure we're delivering excellent quality content.

Download a free PDF copy of this book

Thanks for purchasing this book!

Do you like to read on the go but are unable to carry your print books everywhere?

Is your eBook purchase not compatible with the device of your choice?

Don't worry, now with every Packt book you get a DRM-free PDF version of that book at no cost.

Read anywhere, any place, on any device. Search, copy, and paste code from your favorite technical books directly into your application.

The perks don't stop there, you can get exclusive access to discounts, newsletters, and great free content in your inbox daily

Follow these simple steps to get the benefits:

1. Scan the QR code or visit the link below

https://packt.link/free-ebook/978-1-83588-746-2

1. Submit your proof of purchase
2. That's it! We'll send your free PDF and other benefits to your email

directly

Contents

1. Hands-On Industrial Internet of Things
2. Contributors
3. About the authors
4. About the reviewers
5. Preface
    1. Who this book is for
    2. What this book covers
    3. To get the most out of this book
    4. Download the example code files
    5. Conventions used
    6. Get in touch
    7. Share Your Thoughts
    8. Download a free PDF copy of this book
6. Part 1:Industrial IoT
7. Chapter 1: Introduction to Industrial IoT
    1. Technical requirements
    2. IoT background
        1. History and definition
        2. IoT enabling factors
    3. IoT use cases
    4. IoT key technologies
    5. What is the IIoT?
        1. Use cases of the IIoT
        2. IoT and IIoT – similarities and differences
        3. IoT analytics, AI, and AIoT
        4. Industry environments and scenarios covered by IIoT
    6. Summary

8. Chapter 2: Understanding the Industrial Process and Devices
    1. Technical requirements
    2. The industrial process
        1. Automation in the industrial process
        2. Control and measurement systems
        3. Types of industrial processes
    3. The CIM pyramid
        1. CIM pyramid architecture – devices and networks
        2. CIM networks
    4. The IIoT data flow
        1. The Industrial IoT data flow in a factory
        2. The edge device
        3. The Industrial IoT data flow in the cloud
    5. Summary
9. Chapter 3: Industrial Data Flow and Devices
    1. Technical requirements
    2. The IIoT data flow in the factory
    3. Measurements and the actuator chain
        1. Sensors
        2. Converters
        3. Actuators
    4. Controllers
        1. Microcontrollers
        2. PLCs
        3. Advanced Control
        4. DCS
    5. Industrial protocols
        1. The OSI/ISO model

        2. Automation networks
        3. The fieldbus
    6. Supervisory control and data acquisition (SCADA)
    7. Historian
    8. ERP and MES
        1. The asset model
        2. ISA-95 equipment entities
        3. SA-88 extensions
    9. Summary
10. Chapter 4: Implementing the Industrial IoT Data Flow
    1. Discovering OPC
        1. OPC Classic
        2. OPC UA
        3. The OPC UA information model
        4. OPC UA sessions
        5. The OPC UA security model
        6. The OPC UA data exchange
        7. OPC UA notifications
    2. Understanding the IIoT edge
        1. The IoT edge versus the IIoT edge
        2. The fog versus the IIoT edge
    3. The edge architecture
        1. The edge gateway
        2. The edge computing
        3. The edge tools
    4. Edge implementations
        1. Azure IoT Edge
        2. AWS IOT Edge
        3. Node-RED

    5. Edge internet protocols
    6. Implementing the IIoT data flow
        1. IIoT data sources and data gathering
        2. PLC
        3. DCS
        4. SCADA
        5. Historians
    7. Edge deployment and data flow scenarios
        1. The edge on a fieldbus setup
        2. The edge on OPC DCOM
        3. The edge on OPC-Proxy
        4. The edge on OPC UA
        5. OPC UA on the controller
    8. Summary
11. Chapter 5: Applying Cybersecurity
    1. What is a DiD strategy?
        1. Making people aware
        2. Understanding technology
        3. Operating modes and procedures
        4. The DiD in an industrial control system environment
        5. Firewalls
    2. Common control-network-segregation architectures
        1. Network separation with a single firewall
        2. A firewall with a DMZ
        3. A paired firewall with a DMZ
        4. A firewall with DMZ and VLAN
    3. Securing the IIoT data flow
        1. Securing the edge on a fieldbus
        2. Securing the edge on OPC DCOM
        3. Securing the edge on OPC Proxy

        4. Securing the edge on OPC UA
        5. Securing OPC UA on a controller

    4. Summary

12. Chapter 6: Performing an Exercise Based on Industrial Protocols and Standards

    1. Technical requirements
    2. The OPC UA Simulation Server

        1. OPC UA Server Node.js
        2. Prosys OPC UA Simulator – optional step

    3. The edge

        1. Installing a simple edge client
        2. The Node-RED edge
        3. Node-RED

    4. Summary

13. Part 2:Industrial IoT Architecture
14. Chapter 7: Developing Industrial IoT and Architecture

    1. Technical requirements
    2. Introduction to the IIoT platform and architectures
    3. Microservice, containers, and serverless computing

        1. Docker
        2. Microservice-oriented technologies
        3. Software as a service

    4. The standard IIoT flow
    5. Understanding the time-series technologies

        1. Not only sensors

    6. Asset registry
    7. Data processing and the analytics platform

        1. Excursion monitoring analytics

        2. Advanced analytics
        3. Big data analytics

    8. Hands-on with InfluxDB, Node-RED, and OPC UA
    9. Using Docker Compose and Docker Build
    10. Summary

15. Chapter 8: Implementing a Custom Industrial IoT Platform

    1. Technical requirements
    2. An open source platform in practice

        1. Mosquitto as a data gateway
        2. Storing time-series data on InfluxDB
        3. Connecting InfluxDb and MQTT Data Broker
        4. Starting and testing the data flow
        5. Visualizing data using Grafana
        6. Developing our batch analytics with Airflow
        7. Developing our online analytics with Airflow
        8. What’s still missing here?

    3. Building an asset registry to store asset information

        1. Building an asset model with Neo4j

    4. Pros and cons of the proposed platform
    5. Other technologies

        1. Other open source technologies for analytics
        2. Other open source platforms
        3. Other IIoT data beyond the time series

    6. Summary

16. Chapter 9: Building an AWS Industrial IoT Solution

    1. Technical requirements
    2. Why do we use commercial IIoT platforms?

        1. AWS architecture
        2. AWS IoT
        3. Registering with AWS

        4. Installing the AWS command-line interface client

    3. IoT Core

        1. Identifying the AWS MQTT endpoint
        2. Connecting Node-RED
        3. Using the AWS SDK
        4. Greengrass V2
        5. SiteWise

    4. Summary

17. Chapter 10: Implementing an Industrial IOT Data Flow with AWS

    1. Technical requirements
    2. Architecture of the exercise
    3. AWS IoT Analytics

        1. SageMaker and Athena
        2. IoT Analytics

    4. Storing data
    5. Visualization

        1. Workforce users
        2. Grafana
        3. About visualization

    6. Summary

18. Chapter 11: Performing a Practical Industrial IoT Solution with Azure

    1. Technical requirements
    2. Azure IoT ecosystem
    3. Registering for Azure IoT
    4. Azure IoT Hub

        1. Registering a new device
        2. Building a custom edge
        3. Azure IoT Edge

        4. Device twins
        5. Monitoring messages sent through IoT Hub Metrics

    5. Showing messages sent through Service Bus
    6. Summary

19. Chapter 12: Implementing an Industrial IoT Data Flow with Azure

    1. Technical requirements
    2. Architecture of the proposed exercise
    3. Setting the data flow

        1. Data storage
        2. Data processing
        3. Azure Synapse
        4. Machine Learning analytics

    4. Building visualizations

        1. Azure Data Explorer
        2. Grafana
        3. Power BI
        4. IoT Central
        5. More on Azure Data Explorer

    5. Comparing the platforms
    6. Summary

20. Part 3:Industrial Artificial Intelligence of Things
21. Chapter 13: Performing Diagnostic, Maintenance, and Predictive Analytics

    1. Technical requirements

        1. Jupyter

    2. IIoT analytics
    3. The different classes of analytics

        1. Descriptive analytics

        2. Diagnostic analytics
        3. Predictive analytics
        4. Prescriptive analytics

    4. IIoT analytics technologies

        1. Rule-based analytics
        2. Model-based analytics

    5. Building IIoT analytics

        1. Step 0 – problem statement
        2. Step 1 – dataset acquisition
        3. Step 2 – EDA
        4. Step 3 – building the model
        5. Step 4 – packaging and deploying (MLOps)
        6. Step 5 – monitoring

    6. Understanding the role of the infrastructure
    7. Deploying analytics

        1. Streaming versus batch analytics
        2. Condition-based analytics
        3. Interactive analytics
        4. Analytics on the cloud
        5. Analytics on the edge
        6. Analytics on the controller
        7. Advanced analytics

    8. OSA
    9. Analytics in practice

        1. Anomaly detection in practice
        2. Anomaly detection with unsupervised ML
        3. Anomaly detection with supervised ML
        4. Predictive production analytics in practice
        5. Prescription using language models

    10. Summary

22. Chapter 14: Implementing a Digital Twin – Advanced Analytics

1. Technical requirements
2. Digital twins
    1. Digital twins in practice
    2. Implementing a digital twin in IIoT
3. AI and IoT
    1. ML
    2. DL
    3. GenAI
    4. AGI
4. Developing a digital twin
    1. Preparing the development environment
    2. Evaluating the RUL of 100 engines
    3. Monitoring a wind turbine
5. Platforms for digital twins
    1. Digital twins platforms
    2. Advanced modeling
6. Other kinds of IIoT data
7. Summary

23. Chapter 15: Deploying an Analytics Model
    1. Technical requirements
    2. Understanding model-as-a-service
    3. Developing our first MaaS with MLflow
        1. Starting MLflow
        2. Developing our ML model
        3. Understanding Airflow integration
    4. Working with the Azure ML service
        1. Starting with the Azure ML service
        2. Understanding the Azure ML workspace
        3. Developing wind turbine digital twins with Azure

        ML
    4. Cleaning the resources
    5. IoT Hub integration
    6. Azure IoT Edge

  5. Implementing analytics on Amazon SageMaker

    1. Accessing SageMaker Studio
    2. Preparing an S3 bucket to store data
    3. Developing a digital twin with SageMaker
    4. Consuming the model from AWS IoT Core
    5. Understanding the advanced features of SageMaker

  6. Understanding GCP and multi-cloud solutions
  7. Summary

24. Index

  1. Why subscribe?

25. Other Books You May Enjoy

  1. Packt is searching for authors like you
  2. Share Your Thoughts
  3. Download a free PDF copy of this book

Landmarks

1. Cover
2. Table of Contents
3. Index